KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT DER TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT WERKTUIGKUNDE
AFDELING PRODUKTIETECHNIEKEN
MACHINEBOUW EN AUTOMATISERING
Celestijnenlaan 300B — B-3001 Leuven (Heverlee), België

DEVELOPING ROBOT BEHAVIOURS THROUGH NEURAL LEARNING

Jury :

Voorzitter :
Prof. dr. ir. J. Delrue

Leden :
Prof. dr. ir. H. Van Brussel, promotor
Prof. dr. ir. J. De Schutter, PMA - KU Leuven
Prof. dr. ir. J. Vandewalle, ESAT - KU Leuven
Prof. dr. ir. J.M. Van Campenhout, LEM - RU Gent
Prof. dr. ir. J.P. Kruth, PMA - KU Leuven

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de toegepaste wetenschappen

door

Adhi Sudadi SOEMBAGIJO

U.D.C. 681.3*I29

December 1995

D/1995/5769/13
ISBN 90-73802-51-2

Abstract

This dissertation addresses the use of the neural network learning approach to achieve particular behaviours of robots. It focuses on the development of two chosen behaviours : **robot reflexive behaviour**, which illustrates the on-line skill refinement process and **robot cooperative motion behaviour**, which illustrates the learning-from-example case.

Perception is a part of activities in the robot sensory-motor coordination, where the robot has contact with the environment. Perception involves not merely the reading of sensory inputs but also the understanding of the received patterns. A discussion on robot perception is presented in advance, where the holistic nature of perception is taken as the viewpoint. This discussion is illustrated by the use of a neural network to achieve a holistic **robot touch perception**. An experimental result shows that an LVQ network can be used to classify contact sensations in a robot gripper in a direct, featureless fashion – which reflects the nature of holistic perception.

Robot reflexive behaviour is represented by a visual tracking behaviour, implemented by an eye-in-hand robot arm system. This behaviour can be built through a gradual improvement process based on an immediate reward. A reinforcement-comparison algorithm is tailored to be used with a dynamic model of the environment to control the action explorations. The system configures two back-propagation networks as an action network and a reward predictor network. A system capability to predict the target object trajectory can be built by utilising the property of a time-delay neural network. This methodology is validated both in simulated and real robot environments.

Robot cooperative motion behaviour is illustrated by a control strategy for two cooperative robot arms. The cooperative motion behaviour is developed through an off-line learning by using a set of precollected examples. A fully position-based neural network mapping between two arm configurations is presented, which is called gross-motion mapping. A better representation about the robot interaction is obtained through the sensed contact force between the two arms. A control scheme is proposed to train a neural

network to find a relationship between the contact forces at a certain actual arm position to the compensating joint displacements needed to reduce the positioning error based on only gross-motion mapping. This compensation mapping is called fine-motion mapping. RBF neural networks show their capability to learn both gross- and fine-motion mappings, which are fully nonlinear. The effectiveness of this control scheme is verified in a simulated robot environment.

The presented methodologies and results are aimed at understanding the state of the art and extending the use of artificial neural networks and learning approach in general, in building particular robot behaviours.

Contents

List of Tables

List of Figures

Acknowledgements

The greatest gratitude I would like to express in the first place is to God, **Allah s.w.t.**, for giving me his blessing to attain this achievement, to keep giving me the energy for life and to give the best guidance in my life.

Since this is neither a Nobel Prize nor an Oscar acceptance, the acknowledgements do not need to be short. There are many people to thank for contributing to the five great years I have had at the PMA laboratory.

First of all, I would like to express my great gratitude to Prof. dr. ir. Hendrik Van Brussel for being my promotor and an excellent, enthusiastic motivator. He provides me with a perfect condition to grow in a scientific environment : know-how, challenges, opportunities and financial support. From him, I learned about human aspects as well as academical aspects. He could create a type of relationship that allows me to express confidently disagreements and to argue, as well as to admire and to obey. He also filled me with the consciousness of the senses of quality and beauty.

The most enduring gratitudes go to Prof. dr. ir. Sri Hardjoko Wirjomartono and Dr. ir. Muljowidodo Kartidjo, who ignited my endeavour to attain this doctoral achievement. They have succeeded in changing my mind positively, from "jumping directly from ITB to the family company" to "going a little further to expand the knowledge and see the world"

My education in Leuven could not be accomplished without the scholarship grants I received from firstly, the PMA/IUAP-50 (the Belgian Programme on Inter-University Attraction Poles initiated by the Belgian State - Prime Minister's Office - Science Policy Programming), and secondly, from the K.U. Leuven Research Coordination Office.

My gratitudes to Prof. dr. ir. Joris De Schutter, for giving me – from time to time – few-but-sharp comments and guidance that were very significant to keep my research in the right track. And also for being a member of the reading committee of this dissertation, who could provide me with many invaluable advices.

Unlimited thanks go to Prof. dr. ir. Jean-Pierre Kruth, Prof. dr. ir. Paul Sas, Prof. dr. ir. Dirk Vandepitte, Prof. ing. Paul Vanherck and Prof. Jacques Peters, who have created a convenient atmosphere to work in the PMA lab. My special thanks to Prof. dr. ir. Jean-Pierre Kruth for being a member of my jury.

My thanks also to Prof. dr. ir. Joos Vandewalle for many inputs I received, directly or indirectly, during I followed the enthusiastic activities he organised together with the K.U. Leuven Interdisciplinary Centre of Neural Networks. I was very glad when finally he became a member of the reading committee of this dissertation. Because of that, I received many significant advices concerning my work. Special thanks also go to Prof. dr. ir. Jan Van Campenhout for being a member of my jury.

My warm thanks to Dominiek Reynaerts for initiating me into the culture of the PMA lab. during the early years of my research activities, for being my Master thesis supervisor and for being a research partner. His idea has influenced many parts of my work. Also, thanks for being an officemate during all my five years time, to share everything, including picking-up the telephone-calls, the soft music from the radio, and watering the plants.

Numerous thanks to the PMA's neural-net-workers : Xu Hong, Pan Min-Chun, and Marnix Nuttin. I learned from the working-togetherness we have had that a collaboration, team supports and idea sharing only works if we appreciate each other. Finally, the openness and friendship are the reliable tools to achieve an excellent cooperative atmosphere. Blind competitiveness will only isolate someone. Warm thanks also go to Jurgen Vandorpe for his cooperative personality and support in dealing with mobile robot stuff.

My thanks go to Mark Van Hulle, from the Neuro- and Psychophysiology laboratory, Gasthuisberg University Hospital, K.U. Leuven, for giving the answers of my first stupid questions about neural networks.

Many thanks to the American Robot people : Sabine Demey, Herman Bruyninckx, and Wim Witvrouw, who helped me to get quickly acquainted with the robot, the vision system and COMRADE.

I am grateful to Wim Persoons, who has introduced me with the "real simulated world". By his special introduction, I could quickerly get in with the enjoyful fast-simulation environment. My sweet memory about IGRIP, GSL-language, and Silicon Graphics stuff will link my mind directly to him.

My thanks also go to the students who were working around me during my three years doctoral research time, who were not merely doing the "dirty work" of my research, but also enriched me technically and *non-technically* and who have colourised my way of seeing problems. It is really my pleasure to list them for that : Jan Peirs, Stef Sonck, Frans de Bethune, Phillip Hespel, Luc Hoppenbrouwers and Wim Raspoet. And also a special thank goes to Antonio Jiménez Ruiz from the Instituto de Automática Industrial, Madrid, Spain, who

during his stay at PMA, has been a good research partner of mine, specially in the field of statistical pattern recognition.

Thanks and appreciations to Hans Thielemans, who became my last person to go when I found some desperating technical problems : from the wrong encoder signals until the incorrect camera connection, from the unexpected X Window/Motif appearance until the missing LaTeXfont files.

Many thanks to the DI Personals : Philippe Vanherck and Jan Thielemans, for handling my daily computer problems and for solving various kinds of last minutes unexpected problems with the computers and the network system.

My thanks also go to Lieve Aelvoet, who kept me in touch with the recent literatures and connected me to the outside world sources. I am so sorry for the belated book returns – many times !

Warm gratitudes to Jean-Pierre Merckx, Polleke, Ferno, Remo, Dirk, Eddy, Viggo, and Frans de Kelver, for providing me with excellent expertises that supported the realisation of my ideas.

Technical works could never be performed without administrative supports. For that, my deepest and warmest thanks go to Lieve, Ann, Karin, Carine and Luc Haine. Their extra work during the organising time of the IASE-5 (the fifth conference of the Indonesian Aerospace Students in Europe '93) has placed me in their debt. Special thanks to Lieve for her care and kindness even before I arrived in Belgium.

My never expressed thanks also go to some unforgetable personalities, for their presence during my growth in the PMA lab. and who have given me some hidden lessons :

Dr. Fülöp Augusztinovicz, who always reminded me *not to forget to go home*, if we were the only two people sitting in the computer room late after midnight (actually he forgot to go home more often than me). He is a good example for me of a low profile, highly dedicated scientific person. He was also my first kayak partner, on my first kayak trip in the first PMA week-end I participated in southern Belgium.

Prof. Herman Mann (who is currently at the Czech Technical University, Prague), for the regular tennis practices – even during the Belgian winter time – who always provided me with his time for broadening my horizon with "stories" about his interesting scientific experience.

The late Dr. Simeön Patarinski, who had - implicitly - taught me about self-pride, the sometimes forgotten human characteristic. We shared much time together during his stay at PMA, not only in the office, but also during the bicycle repairing time or a walk between the office and our studios.

Bao Chaoying (who is currently at the University of Western Australia), who unofficially has become my supportive advisor and has given me a lubricant to smoothen my adaptation to the working climate in the lab. during my first year working at PMA.

A colourful time I had also spent at PMA by having Lieven Demeestere – the crazy little thing – as my officemate almost during the whole three years time of my doctoral research. I am very grateful to him for being a supportive friend, and most important for the research-related, thesis-related and life-related talks and consultations we have had. I could be the only one in the lab. who knew about his MTV-taste music. And he has never had the confidence to express his music taste in front of me (most of the time he switched off his music on the radio if I entered the office). He showed almost all the positive sides of a friendship. But sometimes I hated him ! His unmanageable paper stuff most of the time violently occupied parts of my unmanageable desk.

Another nice memory I also got was by being together with other office-mates who have come and gone during my time : Dirk Diddens, Jan Van Hee, Dominiek Surinx and Willem Gejsels. The *most recent type of fun* I got during my last hard occupation of time at the office with my current office-mates : Mark Versteyhe, István Németh, Dominiek Reynaerts and Paul-Henri 's Heeren (who spent his time more in the workshop than at his office). The extended fun-time that I got together with the *week-end explorer* (the week-ends were no longer spent in the lab.) : István Németh and Joan Savall (he is a Valenciano, not a Spanish), has left a special meaning of a friendship.

I had some late nights stimulating talks beside the workstations with José Nájera – the international boy – that has broadened my horizon about the robotics research situation in other parts of the world. He has also introduced me to some important robotics-related information sites in the internet, that I had taken for granted before. Thanks also to Peng Yanming – my second, unofficial system manager – for helping me to get rid of various FAQs (frequently asked questions) about UNIX and LINUX systems. And he, together with Pan Min-Chun and Jonas de Carvalho, became nice mates for sharing the joy of our struggle to meet ourselves-created dissertation deadlines.

The lab. would not be that cheerful without the Flamish hospitality of Luc Bongaerts and the laughter of Pascal Degezelle. I had some nice late afternoon talks also with the changing Jan Detand (he has changed after he returned from Indonesia).

My hormonal desires for organisational activities have been fulfilled by sharing social activities outside the lab. with some great people of Nusa Indah v.z.w. : Gerda, Rita, Mieke, Greet and others. I enjoyed my contributions and involvements in the organising of many Indonesian-culture promotional activities in the Benelux. I have also gained various nice experiences by working together with people of the Indonesian Embassy in Brussel in many social and cultural activities.

I have a great time also with all Indonesian students in Leuven, who, although I had declared that I was in a severe time with my doctorate work, still elected me to be the chairman of the Indonesian Student Association in

Leuven '94-95. I had an enjoyful work together with the Indonesian Student WWW-working group which created the first foreign student association's homepage – linked to the K.U. Leuven homepage. A significant memory I got as well by being together for three up to five years in the PMA lab. with : Bagus Arthaya, Erzi Agson Gani, Indra Tanaya, Tutuko Prajogo, and Indrawanto. Some of them have left special life-meanings for me.

Bagus Arthaya, a hard-worker with an open-minded personality, who never used his time-for-his-family as an excuse not to work hard. Special thanks to him for the important advices for my work in the two-arm coordination.

Indra Tanaya, the big brother, for the behaviour-related discussions and for sharing the dream about the Indonesia-Mechatronics Unlimited Ltd. He has a kind of personality with long-term considerations. Nice discussions are easy to start with him if it is about the future of the Indonesian manufacturing world, the topic that I used to open discussions with him. Also the warmest thanks go to his family for the Indonesian food supplies, almost every time when I was ill. Especially about the dream . . . it is not the end yet . . .

The last but not the least, infinite and incomparable gratitudes go to my parents, the most wonderful couple who brought-up their four children successfully. My father always directed our family to be high achievers. He works harder for and at everything in life than anybody else I have known. I cannot deny to admire his working ethos, vision and spirit. He formed me to believe that working hard is a part of life's recreation. My mother, who is an excellent partner for my father, is the most wonderful mother I could have. She taught all the lessons for life with giving examples, and has superhuman energy and perseverance that inspired everything I have accomplished. From her, I learned the beautiful combination of love, acceptance and hard working. She has taught me a lot about leadership, even since I was in her pregnancy.

My undescribable love goes to my little sister and brothers, who have struggled earlier in the family company, while I was still dealing with robots.

My uncountable thanks also to my family for their distant love and supports and they have always contributed to the creation of my home-sickness.

To them, this dissertation is dedicated.

I realise that by only doing research and without the presence of those people around me, the maturing process of my scientific and social aptitudes in Leuven would be incomplete. And it is sometimes difficult to realise and to count what I have been gaining, and which part of myself has been changing during a five years long process.

But for sure, now I find out that the world is different with the one I saw five years ago.

ADHI SUDADI SOEMBAGIJO
Leuven, December 1995

Symbols and Notations

SYMBOLS

$\triangleq$	The left-hand side is a shorthand notation for the expression on the right-hand side
$\otimes$	Multiplication, used in reinforcement learning algorithms

NOTATIONS

Artificial neuron model

$\mathbf{o}$	Output vector of a layer of neurons
o_i	The state of neuron i
$\mathbf{x}$	Input vector of a layer of neurons
x_j	Input from neuron j in the previous layer
I_i	The set of inputs to neuron i
$\mathbf{W}$	Weight matrix, also called the connection matrix
w_{ij}	The synaptic weight of the connection from neuron j to neuron i. The first subscript i denotes the index of the destination neuron and the second subscript j denotes the index of the source neuron
$f(\cdot)$	Activation function

net_i	Activation value for the i^{th} neuron

Recall and learning computation

$\Gamma[\cdot]$	Nonlinear matrix operator used to map the input space x to the output space o in feedforward networks
t	Time instant
Δ	Delay elements in the feedback loop in feedback networks; if time t is considered as a discrete variable equated to unity, then the time instances are indexed by positive integers, e.g. Δ, 2Δ, 3Δ
$\bar{e}(t)$	Learning information supplied at time t
I_i	The set of inputs to neuron i
d_i	The desired output of neuron i
r	reward in reinforcement learning algorithms; r_{t+1} denotes the future reward
γ^t	The discount rate determining how fast one's concern for delayed reward falls off with length of the delay, at instant t; this is used in reinforcement learning algorithms
$\mathbf{z^{-1}}$	A one step delay, used in reinforcement learning algorithms

Tactile data processing

$\mathbf{F}$	A 3x3 convolution filter used to eliminate noise
$input(x, y)$	The input image of the noise cancellation technique
$fine(x, y)$	The fine image, the resulting image of the noise cancellation technique
i^*, j^*	Pixel row and column indexes (respectively) in the image before preprocessing
i, j	Pixel row and column indexes (respectively) of the preprocessed image
Δy, Δx	The linear displacement values
γ	The rotational displacement value

j_c, i_c	The column and row indexes (respectively) of the contact centroid
x_n, y_n	The column and row indexes of the n^{th} active pixel
A	The number of active pixels
θ	The angle between the minor principal axis of contact area and X-axis of the image
$img(i,j)$	The value of image pixel at (i,j)
$h(i,j)$, $l(i,j)$	Image pixel labels indicate the pixel's groups in the histogram analysis. h refers to the group of pixels with values higher than the specified threshold value and l to the group of pixels with values lower than the threshold

Learning visual tracking

$\vec{X}(t)$	The world state observed by the vision system
$(\Delta x, \Delta y)$	Object's relative position with respect to the robot end-effector
$(\dot{\Delta x}, \dot{\Delta y})$	Object's relative velocity with respect to the robot end-effector
$\vec{ee}_{vel}$	Robot end-effector linear velocity in the horizontal plane
$(\dot{\theta}_1^*, \dot{\theta}_2^*)$	Joint velocity values, as the output of the action network
$(\dot{\theta}_1, \dot{\theta}_2)$	Joint velocity values sent to the robot, after being added by noise
$\vec{u}(t)$	The vector of action at time t, $\vec{u}(t) = (\dot{\theta}_1^*, \dot{\theta}_2^*)$
$\vec{n}(t)$	A vector of random noise values, $\vec{n}(t) = (n_1, n_2)$
F	The dynamic model of the environment
$R^*(t+1)$	The predicted *reward* value
$R(t+1)$	The quality of the resulting world state
δ_{act}	The changes of the relative position between the robot end-effector and the object, resulting from the robot action

$\Delta_{position}$	The Euclidian distance between the robot end-effector and the target object, measured in 2-dimensional plane
$\Delta x, \Delta y$	The object's relative position coordinates with respect to the robot end-effector, observed by the vision system
P_i^{pred}	The input pattern for the training of the prediction network
P_o^{pred}	The output pattern for the training of the prediction network
P_i^{act}	The input pattern for the training of the action network
P_o^{act}	The output pattern for the training of the action network
$\mathbf{x}(t-\tau_m)$	The time-delay network inputs of at time t, where m represents the delay order
$\dot{\theta}_d$	The desired joint velocities output by the neural network controller, expressed in the robot joint velocity controller scheme
$\theta_{d,out}$	The desired joint positions, expressed in the robot joint velocity controller scheme
$\dot{\theta}_{d,out}$	The desired joint velocities, expressed in the robot joint velocity controller scheme
K_p	Position feedback gains (units s^{-1})

Two-arm coordination

$X(t)$	The homogeneous transformation matrix of the object coordinate system with respect to the reference coordinate system at time t
$Y^i(t)$	The homogeneous transformation matrix of the end-effector i with respect to its own base coordinate system at time t
T_r^i	The homogeneous transformation matrix of the base coordinate system of arm i with respect to the reference coordinate system
$T_0^i(t)$	The homogeneous transformation matrix of the end-effector of arm i with respect to the object coordinate system at time t
t_f	The time at which the task is completed

$\Theta_i(t)$	Joint position vector of arm i at time t
$\Delta\Theta_i(t)$	Joint displacement vector of arm i at time t
$\Theta_i^*(t)$	The gross joint position vector of arm i at time t
$\mathcal{N}$	The transformation performed by neural network
$\mathcal{N}_{Gross}$	The gross-motion mapping performed by neural network
$\mathcal{N}_{Fine}$	The fine-motion mapping performed by neural network
K_f	Force feedback gains
$\mathbf{K}_{df}$	Sensor deformation matrix
$\mathbf{K}_o$	Sensor stiffness matrix
$\mathbf{K}_o^{-1}$	Sensor compliance matrix
$\mathrm{F}(t)$	The amplified forces at time t, obtained by multiplying the measured forces at time t by the force feedback gain K_f
$\mathbf{P}_i$	The input patterns for the training of the motion mapper
$\mathbf{P}_o$	The output patterns for the training of the motion mapper

Chapter 1

Towards Autonomous Robotic Systems

1.1 Introduction

"You are saying, I suppose, that we can get the humanoid robots into Earth without trouble", said Dr. Amadiro to Mandamus. "With no trouble at all. There's no question in my mind as to that . . . . "

Dr. Amadiro and Mandamus are two characters in Isaac Asimov's science fiction novel, entitled *'Robots and Empire'*. They are two authorities of *Solaria* who are about to control the *Earth*. Their conversation implies that earth's robotics were undoubtedly under developed, at least compared with *Solaria*.
"We can't do anything about their robotic respect and awe of human beings but that may not give them away . . . .
. . . . the vast spaces between Cities are virtually untenanted except by primitive work-robots . . . . "

IT HAPPENED ONLY IN A SCIENCE FICTION NOVEL. But, if we return back to reality, robotic circumstances on the *Earth* in the preceding decade more and less matched to the circumstances Asimov depicted in his novel.

When the "first generation" of industrial robots dominated the field through the 1970's and 80's, robotics was nearly synonymous with manipulators, and often with automated guided vehicles (AGVs). These robots dealt mostly with eliminating many hazardous jobs and performing simple, repetitive tasks with precision and reliability. Such systems are now highly developed and reliable components of modern manufacturing enterprises. However, in general, industrial robots are "dumb". They do not have significant cognitive ability; they cannot operate autonomously. They are "just" preprogrammed to perform specific tasks.

However, since the early days of robot applications, sensors have improved and computers have become smaller, more reliable and much more powerful. With the availability of these tools, the time is ripe for a new phase in the development and application of robots and it is the time for autonomous robotic systems.

Robots can now become more intelligent and autonomous, capable of performing a wide variety of tasks with little aid and supervision by humans. This situation has attracted a new approach for developing robots, which is mainly based on the concepts of autonomy and intelligence. As autonomous systems, robots must be independent systems that exhibit particular interaction with the world, such as manipulation or locomotion. They are different from simple automatic systems which are *self-regulating* but do not make the laws that their regulatory activities seek to satisfy. Autonomous systems develop for themselves the laws and strategies according to which they regulate their behaviour : they are *self-governing* as well as self-regulating (Smithers, personal communication, September 1992, as cited in [Steels, 1995]).

Robots are said to become intelligent as well. When are robots called intelligent ? One working definition defines an intelligent robot as a machine that can extract information from its environment and use knowledge about its world to move safely in a meaningful and purposeful manner [Arkin, 1995]. Although some researchers insist that a robot must possess additional attributes such as world model maintenance, planning, and learning capabilities, intelligence is finally a robot's capacity for active participation in its world. Indeed, without a thorough understanding of a robot's relationship with its environment, it is impossible to effectively construct an intelligent robotic system.

Perception and action are parts of activities by which robots have contact with the world. Perception involves interpreting sensory inputs. Action includes the motor ability to manipulate objects or to navigate through the world. The robot sensory-motor integration requires the construction of a map relating the stimulating sensory patterns to the appropriate motoric commands. In order to build robots that live in the world, these sensory-motor coordination processes must be well understood.

The way how a system coordinates its sensory-motor activities reflects the behaviour of that system. The problem is that it is not always easy and reliable to program the robot behaviour in advance and to represent the world as an a priori knowledge. In the changing environment, system adaptivity, or even evolution, becomes very important, especially in situations where the robot–world interaction cannot be explicitly represented.

Robot learning has emerged as an potential concept to build intelligent autonomous robots. Why ? It is worth looking at how humans and other higher biological organisms undertake their sensory-motor coordination. Hu-

man beings develop and refine their sensory-motor coordination skill through practice, which is in fact learning from experience. This feature leads humans and other higher biological organisms to be so superior to our present technical solutions of the sensory-motor control. The main characteristic of biological sensory-motor coordination is that it is not rigidly preprogrammed, the environment stimuli and system responses are not explicitly represented, but rather, adapt and develop in a maturing phase by concerted actions of sensory and motor experience.

Another goal of learning in a robot is to prepare it to deal with unforeseen situations and circumstances in its environment. Learning may be equally important in the construction of robust robots. The saying *"it can only do what it was programmed to do"* no longer applies, as the robot must literally be programmed to do more than it was programmed to do [Brooks and Mataric, 1993].

On the other hand, building robots that learn to perform tasks has been acknowledged as one of the major engineering challenges. "Why do it when we currently do not understand completely yet ?" The answer is that we believe that some of the insights that result from studying robot learning may not be obtainable any other way. Robot learning forces us to deal with the issue of integration of multiple component technologies, such as sensing, planning, action, and learning itself. While one can study the learning component in isolation, studying them in the context of an actual robot can clarify issues arising in the interface between the various components. That perspective has inspired the author to work out this dissertation. The main focus is to use learning techniques for developing robot behaviours, which are in particular believed to be better developed through non-representational ways. Dealing with robots and make them learn is certainly not an easy job. The method to perform the learning itself also becomes an important issue.

On the other side of the wall, people are busy with the development of Artificial Neural Networks (ANNs), a promising biologically inspired learning technique which exploits the massively parallel processing and distributed representation properties that are believed to exist in the human brain. The primary intent of ANNs is to explore and reproduce human information processing tasks such as speech, vision, olfaction, touch, knowledge processing and motor control. As a matter of fact, robots and ANNs have something in common. They both try to imitate certain aspects of human beings. Robotics mainly approaches the aspects of human motion while ANNs approach the aspects of human mind. Combining them can be a beautiful endeavour towards building intelligent autonomous systems.

As for a doctoral research, not all aspects of this combination are "feasible" to be done by one person. Limitations are sometimes also beautiful. A selection of some relevant problems and test cases, somehow, must be made. Since developing robot behaviours using ANN learning is the main focus

of this research, the selection of the ANN learning paradigm for particular robot behaviours has to be done with knowledge of the expected robot behaviours. A number of behaviours are investigated in the test cases. The author views these behaviours more from the human-like-behaviour point of view, although no attempt is made in this research to relate robot learning techniques in a formal way with psychological models; but rather, analogies with human learning are cited informally.

Learning approaches are very different from the classical engineering approaches which more rely on explicit representations. It needs a different point of view to *believe* how learning techniques can be a promising alternative. The following sections describe issues and perspectives that expectedly could bring the reader to the "right" point of view to start understanding the main content of this dissertation. These will be ended up by a section which describes the dissertation structure.

1.2 Imagine How We Would Do It . . .

In our daily activities, we unconsciously perform an infinite number of actions as back-up for conscious actions. There are countless things to which we pay absolutely no attention. Or, more explicitly, we do not consciously control or monitor many vital actions.

Imagine that we want to take a mug located on a table in front of us. While approaching, reaching and picking the mug, we certainly do not calculate the accelerations involved in moving our body, nor do we consider the Coriolis and centripetal accelerations, the flexibility and deformability of our limbs, our resonance frequency, the friction between our hand and the mug, or whether we might avoid obstacles and collisions. Yet we certainly know how to reach and grasp the mug . . . and we almost never fail.

What do we do to succeed ? In principle, we "know" our dynamics, not the quantitative aspects, but the qualitative, heuristic and approximate version of it. We learned it by experience. We know how fast we have to react to avoid a potential problem. And generally speaking, we do not wait for the last moment, but react as soon as we are aware of the fact. Our knowledge did not come to us through mathematical analysis, but through our daily experience of doing the same actions — our learning capabilities and accumulated experience.

1.3 Knowledge Acquisition versus Skill Refinement

Human beings perform two basic forms of learning : **knowledge acquisition** and **skill refinement**. Knowledge acquisition is defined as learning new

symbolic information coupled with the ability to apply that information in an effective manner. In this case, the essence of learning is the acquisition of new knowledge or significant concepts, including descriptions and models of physical systems and their behaviour. It incorporates a variety of representations, from simple intuitive mental models, examples and images, to completely tested mathematical equations and physical laws. An example of this learning type is someone learning physics. The pupil acquires significant concepts of physics, understands their meaning, and understands their relationship to each other and to the physical world.

The second kind of learning, called skill refinement, differs in many ways from knowledge acquisition. Whereas the essence of knowledge acquisition may be a conscious process whose result is the creation of new symbolic knowledge structures and mental models, skill refinement occurs at a sub-conscious level by virtue of repeated practice. Consider the human ability of riding a bicycle, making a smashing tennis backhand or crossing a busy street. These skills are usually not developed based on significant descriptive concepts. This kind of learning is empiric in nature and based on experience.

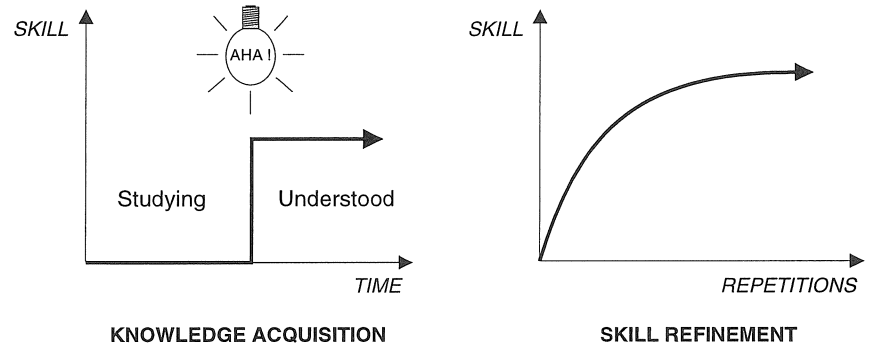


Figure 1.1: The different of two basic forms of learning : knowledge acquisition and skill refinement, in acquiring knowledge.

Figure 1.1 illustrates what is meant by the two kinds of learning. In performing knowledge acquisition, after a certain time, knowledge is acquired and experience is accumulated in order to analyse a particular problem. Suddenly, often after a substantial effort and analysis of causal relationship, the critical break-point is reached, marked on the figure by "AHA !". At that moment, the problem solution is found, and moreover, the solution seems obvious. In performing skill refinement, knowledge is gradually acquired, during the practice and by the accumulated experience. In this case, it might be that a saturation is reached if one remains too long active in one particular field. Most human learning appears to be a mixture of both activities.

1.4 Planning and Action

To act "intelligently", autonomous robots need to be equipped with powerful planning and control capabilities. Planning is an off-line activity that entails reasoning upon an explicit representation of the environment and the task to be performed. Control is an on-line activity, where an actual motion is made to adjust as closely as possible to a commanded motion through the use of sensory feedback.

Suppose now that the task of reaching the mug mentioned above has to be performed in a situation where the mug is placed among other objects on the table and, in particular, sitting behind a bottle. This can be done in two stages. In the first stage, we will think and find out a rough path to reach the mug and, in particular, a decision is made concerning the side of the bottle from which we will approach the mug. Once a path through the space is planned at this coarse level, a second stage of "processing" is entered which is now to "send" a command to our arm to move following the path previously decided, while avoiding collisions with obstacles in the way to the mug. Again, we do not need to "calculate" how should our hand movement be to overcome a potential problem, even though in an environment that may be (partially) unknown a priori and dynamic. Our reaction to avoid the potential problem during the way is performed merely by regarding to the "local" situation. We do not include this reaction in the planning stage.

The essential trait of planning is precisely that it takes place before the actual execution of movements, thus entailing the capacity to generate and manipulate internal representations of both the movements and the world. On the contrary, in the control of motion there is no need for an explicit representation of movement, but on-line monitoring of some sensory variables is required. In human beings, the former is done at a conscious level and requires the concourse of central cognitive functions, while the latter is unconscious and is carried out in a peripheral distributed way.

Thus in robotics, global planning seems amenable to the type of processing which deals with symbolic representation and processing, that allow to perform reasoning processes, whilst local reactive action is more appropriate for the type of processing that can provide a fast and direct mapping between sensory stimulations to motoric actions.

1.5 Computers versus Brains *

So far, we have referred several times to the excellence of human sensory-motor capability. And we believe that today's computer technology has at-

* Some parts of this section are adopted from [Kent, 1981].

tained a certain point where speed, size and price are no longer mutually exclusive. But, why intelligent machines have never been realised ? Even though we know that both are machines and both are information-processing machines, but why then are we having so much trouble making our computers behave in brain-like fashions ? Might an old adage be right that a computer equal to the human brain would require a machine with the size of the Empire State Building, and with the electrical output of Niagara Falls to power it. No, it might not. It was always just an excuse for an unrealised idea.

As a matter of fact, the brain's architecture is quite different from that of a computer, and these differences are very instructive with regard to the task of building a "thinking machine". The brain and the computer have both developed in an evolutionary manner, with "survival of the fittest" determining what features were retained and which were discarded. Their designs differ because nature and computer engineers have different notions of what constitutes fitness. There are two aspects of this difference : the problems the machine is required to solve and the hardware available to build the machine. The successful brains, whose genes contributed to the next generation, were those that were able to solve problems such as recognition, avoiding predators, and finding food. Ability at higher mathematics was never a very important criterion in determining successful brain design, and our poor brains get quickly strained when they are required to do much of it. Primarily, computer design was judged in terms of its ability to perform mathematical functions. As a result, computers are successful at mathematics, but are failures at finding food.

The hardware available to computer engineers and to organic evolution was also different and this has determined in part the differences in the architectures of successful brains and successful computers. Although, as they are seen, the logic gate and neuron have a great deal in common, some of their differences have turned out to have far-reaching consequences. The brain does not have speed on its side – neurons operate in milliseconds not nanoseconds – but it never lacks for quantity. On the other hand, computer engineers were limited in quantity because of the expense and difficulty of assembling components, which dictated designs that were hardware-conservative. This was compensated by using speed to substitute for quantity. Thus, our computers have emphasized small bytes and few registers, but achieved high data throughout with iterative reuse of these components at great speeds.

In contrast, parallel, multiprocessor designs with hierarchical organisation, which the brain uses with wild abundance, are seen in comparatively primitive form in our current computers and computer networks. As presently configured, our computing machines are terribly inefficient at the kinds of problems brains solve easily, and prodigious feats of programming, vast amount of memory, and all the speed that can be mustered give us only the most triv-

ial results. On the other hand, brains have also found an optimal architecture for the kinds of processing problems which they face. This suits them very well to certain activities such as those involving concurrent manipulation of large amounts of data, real-time operations, and finding adequate solutions to complex problems as opposed to exact solutions to simpler ones.

Thus, the brain and the computer have each found a design best suited to the problems they are required to solve and to the hardware available, albeit large brains can perform some computer-like functions poorly, and large computers can perform some brain-like functions poorly. The computer has developed an architecture optimised for logical and mathematical problems, rather than for the display of basic common sense.

The understanding of these differences may bring ideas for the creation of new approaches to robot sensory-motor coordination, which should be not restricted anymore by the fashion of the earlier developed approaches.

1.6 Biological versus Artificial Neural Networks

The basic unit of brain structure is called a neuron. It is a cell body, which is specialised in information processing. It can be thought of as performing the same function as a logical gate in a digital machine or an operational amplifier in an analog machine. The human brain is constructed of billions of neurons.

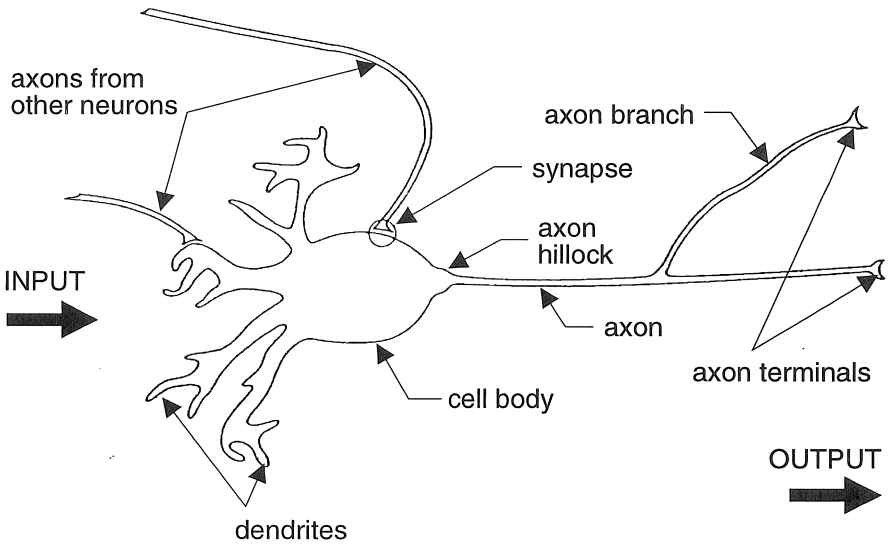


Figure 1.2: The principle parts of a biological neuron [Kent, 1981].

As seen in fig. 1.2, the output of the neuron appears on the long thin part labeled *axon*. The axon can be thought of as a wire. The brain uses it for transmitting information over a distance. The difference with a wire in a digital machine is that axon transmits only pulse streams, not direct current levels. It is a digital wire. All the pulses are approximately the same height and duration and can be thought of as binary bits. Only 1s and 0s are allowed. Another part called *axon hillock* is a Schmitt trigger. It puts a pulse on the axon whenever the analog voltage in the part labeled *cell body* exceeds a preset threshold value. Whenever this occurs, the voltage in the cell body (i.e. the potential difference between the interior and the exterior of the cell) is reset to the baseline or initial value. This voltage rises toward the Schmitt trigger's threshold, or falls away from it, by the action of pulses impinging on the cell body from the axons of other neurons. The place where an axon meets a cell body is called a *synapse*, an it transmits only in one direction. It can be thought of as a diode. The effect of activity at a synapse may be positive or negative; that is, a pulse at a given synapse may either add to or subtract from the cell body voltage. This is a function of the synapse which transmits the pulse. The voltage in the cell body is an analog voltage and represents an algebraic sum of the inputs. The inputs may have different weights, and are placed on an extension of the cell body called a *dendrite*. When the summed inputs to the cell body exceed the threshold of the axon hillock, and a pulse is placed on the axon as a result, it is said that the neuron has "fired".

In artificial neural networks, a neuron is modelled as shown in fig. 1.3 (according to the model proposed by McCulloch and Pitts, 1943).

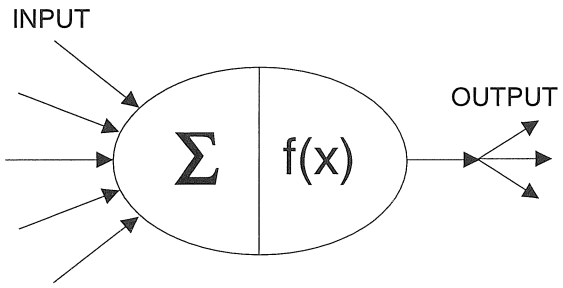


Figure 1.3: The model of an artificial neuron.

It can be considered a threshold unit – a processing element that collects inputs and produces an output only if the sum of the input exceeds an internal threshold value. As a threshold unit, the neuron collects signals at its synapses and adds them together. If the collected signal strength is great enough to exceed the threshold, a signal is sent down the axon which abuts other neurons

and dendrites. The cell body adds all signals, gated by the synapses from the impinging dendrites. The total signal is then compared to an internal threshold value of the neuron, and propagates a signal to the axon if the threshold is exceeded. The neuron internal threshold value, in its extended form, is represented by activation functions. Some kinds of linear, step, ramp and sigmoid-like functions, as depicted in fig. 1.4 are normally used. Artificial neural networks are created by interconnecting many of the simple neuron models into a network.

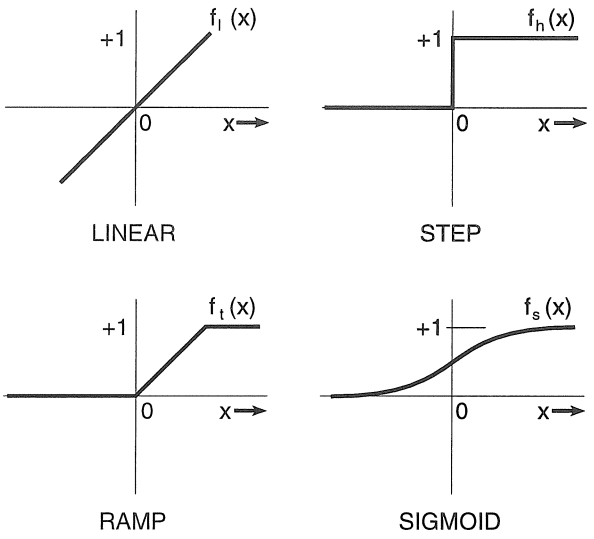


Figure 1.4: Activation functions. Besides between 0 and 1, some neural network paradigms uses activation functions with outputs between -1 and 1.

The artificial neurons have many of their features much more simplified than ones of the biological neurons. Examination of the electrical behaviour of a biological neuron reveals that the greater the input to the cell body, the more quickly the analog voltage will reach the threshold, and the more quickly a new pulse will be placed on the axon after the reset following the preceding pulse. The biological neurons use frequency coding to indicate the magnitude of the summed input activity. Sequential pulses on the incoming axon can sum with one another to produce a greater analog voltage. This voltage is proportional to the frequency of the incoming pulses and thus transforms frequency-coded input back to the analog mode.

Interaction of inputs from different axons is termed "spatial summation", and interaction of sequential pulses on the same axon is term "temporal sum-

mation". One of the interesting things that the brain can do with this capability is to use both place and frequency coding simultaneously on the same line.

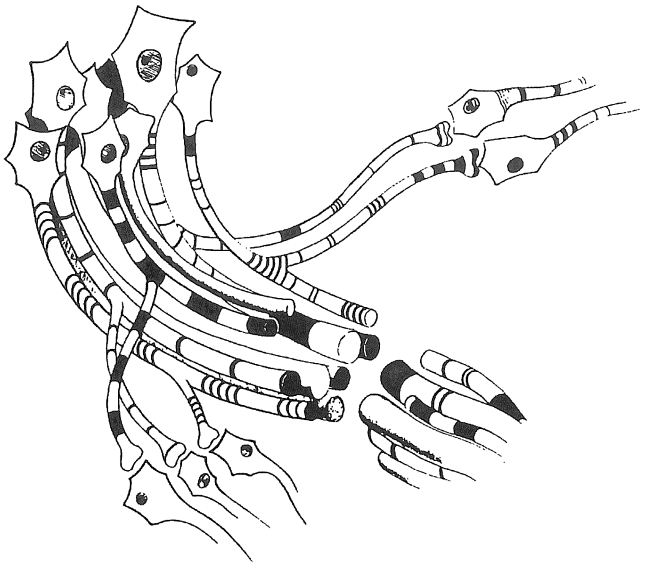


Figure 1.5: The "2-dimensional byte" of the neural bus structure. The active axons carry intensity information, which is seen in this figure as the pulse frequencies indicated by the spacing of the dark areas [Kent, 1981].

Another advantage is the ability to represent very large numerical quantities with what might be called a "temporal byte", or the integration over a brief time period of a single input line. As an example, consider how the brain encodes sensory information from your skin. The type of sensation (i.e. heat, cold, pressure, etc) as well as the location of the sensation is place-coded. That is, *what* we feel is a function of which axon is active. The magnitude of the sensation, *how much* we feel, is frequency coded on the same line. Thus, the brain structure receiving the information can determine the type and location of the stimulation with a "spatial byte" (place code), which determines the set of active lines, and the intensity of the stimulation with a "temporal byte" of frequency code.

The brain's basic "byte" has both a spatial and a temporal dimension. Two independent sets of information can be encoded in these two dimensions. Figure 1.5 illustrates this 2-dimensional scheme of the neuron information en-

coding system. Notice that the spatial aspect of the byte is essentially digital information, and that the temporal aspect of the byte is essentially analog information, although it is encoded in the frequency of digital pulses.

So far, there has not been any ANN paradigm yet translating this scheme into its artificial version. The mathematical treatment of this scheme would be a great challenge for information theorists.

ANNs exhibit many characteristics which offer potential advantages to the building of robot intelligence. This has led to focus this dissertation on ANNs as the technique to be applied.

1.7 Dissertation Overview

Different issues and perspectives have been discussed with an expectation that by composing them, a "right" point of view to start can be obtained. Two learning applications to develop two different robot behaviours : **learning reflexive action** and **learning cooperative behaviour** are the main parts of this dissertation. Those are discussed respectively in **chapters 5** and **6**. Two introductory chapters, **chapters 2** and **3**, precede to provide the state of the art of robot sensory-motor coordination, and of learning and ANNs, followed by an extended discussion about perception which is presented in **chapter 4** .

According to their order, the chapters are described more detailedly as follows :

Chapter 2 : *"Sensory-Motor Coordination : Neuronal or Symbolic ?"* is intended to present a convincing argument of a new vision to sensory-motor coordination. In this chapter, the author's thesis about how the robot sensory-motor coordination should be treated, which is based on that vision, is presented. This deals with a basic argument to build systems that can integrate perceptual inputs smoothly with motor responses, which will bring a major distinction on the development of internal representation to model the environment and to relate sensory and motor information. An intensive literature survey, from engineering and biological point of view, is provided to be used as the basis for underpinning the author's thesis.

Chapter 3 : *"Learning and Neural Networks in Robot Control"* presents the state of the art of robot learning and the application of ANNs in robotics. A literature survey of some important ANN learning paradigms and control schemes are presented.

Chapter 4 : *"Tactile Perception as a Holistic Process"* presents some aspects of machine perception. It starts by presenting some psychological background on human perception. From there, a connecting line is drawn

to find some relationships with its artificial counterpart : robot perception. This is strengthened by an experiment using robot tactile sensors, which imitate the human touch sensing system.

Chapter 5 : *"Learning Reflexive Action : Connecting Vision to Motion"* and **Chapter 6 :** *"Learning Cooperative Motion Behaviour"*, the main parts of this dissertation, describe the two ANN learning applications.

Chapter 7 : *"General Conclusions"* ends up all the preceding chapters with conclusions and directions for future developments.

Chapter 2

Sensory-Motor Coordination : Neuronal or Symbolic ?

It is a profoundly erroneous truism . . . that we should cultivate the habit of thinking what we are doing. The precise opposite is the case. Civilisation advances by extending the number of important operations which we can perform without thinking about them.

PHILOSOPHER ALFRED NORTH WHITEHEAD

2.1 Introduction

Robots are physical devices constructed to imitate aspects of human or animal behaviour which involve interaction with the world. The fashion of the engineering computation over the last thirty years, which has had a strong influence on aspects of single-processor computer architecture, has led most of the classical techniques to approach the robot sensory-motor coordination rather by modularising perception, world modelling, planning and action execution.

In fact humans and animals never behave in that way. Inspired by how humans and animals undertake that task, new approaches to the robot sensory-motor coordination have emerged, based on some aspects of biological system principles. These try to build systems which can integrate perceptual inputs

smoothly with motor responses, even in the presence of novel stimuli and changes in the environment. A major distinction of these approaches is on the development of internal representation to model the environment and to relate sensory and motor information.

2.2 The Classical Engineering Approaches

A control system for a completely autonomous robot must perform many complex information processing tasks in real time. It may operate in an environment where the conditions are changing rapidly.

In essence, a classical control technique entails that the complete structure of the mechanism and its interaction with the environment be solved in advance, and expressed in the code of explicit mathematics and logic.

2.2.1 Problem Decomposition

There are many possible approaches to building intelligent systems. As with most engineering problems, they all start by decomposing the problem into pieces, solving the subproblems for each piece, and then composing the solutions. Typically, robot builders have sliced the problem into some subset of :

- sensing
- mapping sensor data into a world representation
- planning
- task execution
- motor control.

This decomposition can be regarded as a horizontal decomposition of the problem into vertical slices [Brooks, 1986] as illustrated in fig. 2.1. The slices form a chain through which information flows from the robot's environment, via sensing, through the robot and back to the environment, via action, closing the feedback loop (of course most implementations include internal feedback loops also). An instance of each piece must be built in order to run the robot at all. Later changes to a particular piece (to improve it or extend its functionality) must either be done in such a way that interfaces to adjacent pieces do not change, or the effects of the change must be propagated to neighbouring pieces, changing their functionality too.

Presently, there are two classical approaches for the solution, characterised by the single-processing computational model. One is the built-in-model based

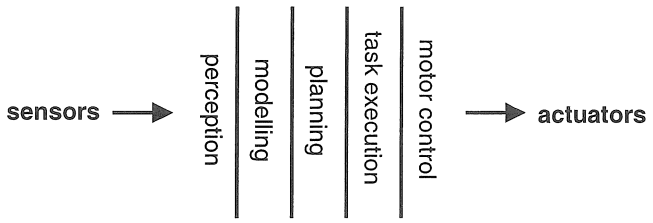


Figure 2.1: Traditional horizontal decomposition of a robot control system into functional modules [Brooks, 1986].

approach. This approach is devoted to rigorous mathematical analysis of the kinematics and dynamics of the mechanisms, and to the development of analytical schemes for their control. The other one is the traditional artificial intelligence (AI) approach. This approach relies on a cognitive process, which is based on the manipulation of the symbolic representation of the world and ideas.

2.2.2 Issues in Built-In-Model Based Approaches

In dealing with the physical interactions with the world, the built-in-model based approach is devoted to the representation of the interactions by means of complete advance knowledge. Methods have been developed within this approach for general robotic tasks, which generally can be grouped into two major classes : methods based on **off-line planning** and methods based on **reactive control**.

Off-line planning methods combine geometric analysis of the robot-world configuration with physical analysis of the tasks to determine motion strategies that will result in successful execution. A number of interesting developments can be found in the domain of peg-into-hole insertion, a benchmark task which is commonly used for compliant motion control under conditions of uncertainty [Whitney, 1982, Gustavson, 1985]. In the presence of uncertainty in sensing and control, some researchers have suggested incorporating the uncertainty into the model [Lozano-Pérez et al., 1984, Erdmann, 1986]. [Bruyninckx, 1995] handles the kinematic model for force controlled compliant motion by applying an on-line identification of geometric uncertainties.

Another interesting domain is the local motion planning in the navigation of autonomous mobile robots[1]. Techniques have been developed by a num-

[1]There are two types of planning in mobile robot control : global path planning and local motion planning. Global path planning deals with planning paths towards the goal positions, based

ber of researchers to handle both global and local planning, such as visibility graphs [Lozano-Pérez, 1981], potential fields [Khatib, 1986], and modelling the free space [Brooks, 1983, Giralt, 1984]. The world of a mobile robot is the internal representation of the robot's external world. This is a database in which the knowledge about the world is stored in the form of geometrical relations. The applied techniques take advantage of the accumulated knowledge from the environment. They need to keep the world model to be always up-to-date. The variation of the environment should be adapted into the model in the most efficient way. On the other hand, in most cases, the sensory information extracted from the environment is uncertain and not fast enough for the real-time performance of the local motion planning.

Moreover, any theoretical representation of the real world requires practically a large number of simplifications, and these will be supported by a large number of assumptions about what are the relevant factors to represent, and what constitutes an adequate way to represent them.

As robotics adopts the idea of having a complete world model, a number of subproblems arise, such as that perfect models of the world cannot be obtained from sensors, or even from CAD-databases. Furthermore, internal models of the robot system itself are often built with a lack of a priori knowledge of some intermediate transformations (e.g. robot dynamics, sensors-to-world transformations, etc.). Finding these transformations is sometimes not an easy task. Once this is done, the robot can be controlled very accurately. However, this is generally a difficult and computationally intensive task.

Up to this point, a reasonable question may arise : "Is having the complete world model, the robots' prerequisite towards behaving intelligently ?" Therefore, much work then has been devoted to reactive control methods. These methods try to counter the effects of uncertainty with on-line modification of the motion control based on sensory feedback. In [De Schutter, 1986], a remarkable work has been achieved for compliant motion control by incorporating a force/torque sensory feedback loop around a position control loop. This work was achieved after a long research experience in compliant motion in the same research group, performed by [Van Brussel and Simons, 1979] with the active adaptable compliance wrist (AACW) concept and with the stochastic automaton theory by [Simons et al., 1982]. A motion and force control of robot manipulators based on a so-called operational space control architecture is proposed in [Khatib and Burdick, 1986]. Most of the cases in reactive control for compliant motion incorporate force or tactile feedback which comes from only one sensor.

Problems arise when the same technique is applied in a system which use

on the complete world model. On the other hand, local motion planning deals with overcoming the situations nearby the robot during the execution of the planned path, for example to avoid obstacles.

multiple sensory information, such as in a mobile robot. In some situations, redundant sensors are even needed to increase the accuracy with which a robot perceives its surroundings [Flynn, 1985]. Besides the time required by several individual sensor data processing steps, the integration of that sensory information in the control loop mostly needs some additional fusion techniques. Any additional technique practically takes another processing time that causes significant time delay before the reactive control is able to compute a new execution decision [Vandorpe and Van Brussel, 1994]. The time delay problem also arises when a complex sensory interpretation has to be done to achieve a good world perception, such as interpretation of visual information. Such interpretation often invites errors due to imprecise calibration procedures and limitation of the applied models [Wallace et al., 1985].

2.2.3 Issues in Traditional AI Approaches

Traditional AI has tried to tackle the problem from the top down. It tackled intelligence through the notions of *thought* and *reason*. There are things human beings only know about through introspection. The field of AI has adopted a certain *modus operandi* over the years, which includes a particular set of conventions on how the inputs and outputs to thought and reasoning are to be handled (e.g., knowledge representation), and the sorts of things that thought and reasoning do (e.g., planning, problem solving, etc.). It is argued that these conventions cannot account for large aspects of what goes into intelligence [Brooks, 1991a, Ritter et al., 1992].

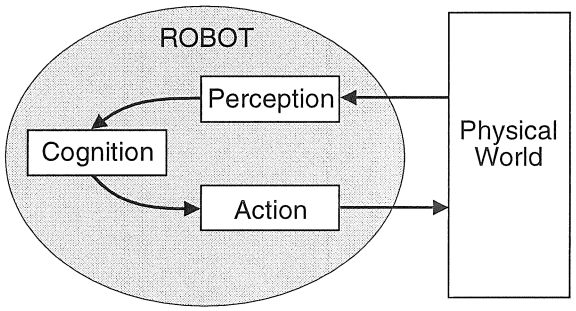


Figure 2.2: AI view to the sensory-motor robot control system [Rich and Knight, 1991].

In viewing sensory-motor robot control systems, the traditional symbolic AI is mostly concerned only with cognition as the main part of the intelligent program development (see fig. 2.2). By this view, cognition is disembodied.

Sensors and effectors are then simply added to the program. Sensors supply the representation of the world, and this representations are processed, resulting in additional representations that correspond to instructions to be sent to the effectors. But in fact problems in perception and action are substantial in their own right.

In the past, robotics and AI have been largely independent endeavors, and they have developed different techniques to solve different problems. Such way of development has led to several implications :

- The input to an AI program is symbolic in form, while the input to robot is typically an analog signal.
- The "pure thought as symbol manipulation" view was applied not only to problems that are essentially symbolic manipulation in form, such as theorem proving, but also to problems that had to do with the robot's own body, such as planning movements of its body parts, or to phenomena in which sensory interaction with the world was very important. Some researchers have suspected that perhaps the problem was being made more, rather than less, complex by emphasising the traditional centralised symbol-processing models.
- Robot sensors are inaccurate, and robot effectors are also limited in precision. There is always some degree of uncertainty about exactly where the robot is located and where objects and obstacles stand in relation to it.
- Many robots must react in real time, thus require continuous monitoring and reacting. On the other hand, AI heuristic search is time-consuming. It cannot afford to generate optimal solutions ahead of time.
- The real world is unpredictable, dynamic and uncertain. A robot cannot expect to maintain a correct and complete description of the world. This means that a robot must consider the trade-off between devising and executing plans. This trade-off has several aspects. For one thing, a robot may not posses enough information about the world for it to do any useful planning. In that case, it must first engage in information-gathering activity. Furthermore, once it begins executing a plan, the robot must continually monitor the results of its actions. If the results are unexpected, then replanning may be necessary.

Although the old idea of simply attaching sensors and effectors to existing AI programs has given way to serious rethinking of basic AI techniques in dealing with the physical world [Rich and Knight, 1991], one might ask whether reasoning is essential for mapping perceptions onto appropriate actions.

2.3 A New Vision to the Sensory-Motor Coordination

Driven by the need of robots for dealing with the real world, and concerned that the complexity of run-time modelling of the world is getting out of hand, it seemed that intelligence of being responsive to dynamic aspects of the environment is a reasonable requirement. Robots are supposed to operate on time scales similar to those of humans and animals. The required intelligence should be able to generate robust behaviour in the face of uncertain sensors, an unpredictable environment, and a changing world. Rather than modularise perception, world modelling, planning and execution, the entailed approach should be going towards the smoother mappings between perception and action, which is capable of handling the real time issues involved with interacting with the world.

Acknowledgements of that vision have come from three directions :

- Non-Representational Approaches
- Reactive Approaches
- Distributed Approaches

2.3.1 Non-Representational Approaches

Perhaps much of intelligent action does not require or use explicit representations and their processing. Look at how human beings would undertake many actions in daily activities. Human beings do not consciously control or monitor many vital actions. It seems that many intelligent actions in living systems are not generated by a sequence of information processing steps. Instead, they are likely to be generated by a short connection between sensors and actuators.

Agre and Chapman [Agre and Chapman, 1990] claimed that most of what people do in their day-to-day lives is not problem solving or planning, but rather it is routine activity in a relatively benign – but certainly dynamic – world. Furthermore the representations a robot uses of objects in the world need not rely on naming those objects with symbols that the robot possesses, but rather can be defined through interactions of the robot with the world.

Brooks [Brooks, 1990, Brooks, 1991b] pointed out that internal world models that are complete representations of the external environment, besides being impossible to obtain, are not at all necessary for robots to act in a competent manner. Many of the actions of a robot are quite separable. Coherent intelligence can emerge from independent subcomponents interacting in the world.

Much of the work in this direction made use of learning methods to represent knowledge of the world and its interaction with the robot. Artificial neural networks (the broader discussion will come in chapter 3) has been embraced warmly by many researchers, since they can provide such a non-representational account of cognition.

Many successful results using ANNs have been published. Most of them put emphasis on the non-representational complex model of robot. In the field of hand-eye coordination, ANNs demonstrate the ability to self-organise the relationship between the robot arm and the view of a stereo camera fixed in the world [Grossberg and Kuperstein, 1989, Ritter et al., 1992, Lee et al., 1992]. Other demonstrations show that ANNs can learn to map the appropriate robot arm motion based on image perception obtained from an eye-in-hand camera [Hashimoto et al., 1992b, Torras et al., 1994]. One spectacular, on the road demonstration has been achieved in the field of mobile robot guidance, where ANNs can learn to map directly the camera images of the road ahead of the vehicle, to the steering directions of the wheels [Pomerleau, 1989].

However, it is argued in [Chandrasekaran et al., 1988] that artificial neural networks are as representational as the traditional symbolic systems. The major difference is on the type of representation.

2.3.2 Reactive Approaches

For many tasks, the appropriate architecture for producing solutions is the one that is **reactive**, i.e. the responses are indexed directly over the situation description, rather than resulting from complex problem solving using abstract world models. Most actions are indexed directly by sensory abstractions. As actions are taken, the changes in the world are monitored directly and additional steps are taken reactively as well.

Some work has demonstrated the successful implementations of robot reactive behaviours. In [Dubrawski and Crowley, 1994], an ANN controller has been used for a mobile robot reactive navigation in an unknown, cluttered environment. [Xu and Van Brussel, 1995] implements an ANN reflexive avoidance behaviour of a mobile robot, based on the raw sensor data from ultrasonic sensors. Another demonstration in [Berns et al., 1992] has shown that ANNs can learn on-line to track reactively another mobile robot. Reactive robot controllers for compliant robot motion of peg-into-hole assembly insertion have also been demonstrated by using ANN learning [Asada, 1990, Gullapalli et al., 1992, Nuttin et al., 1995].

2.3.3 Distributed Approaches

This direction involves some aspects of non-representational and reactive approaches, but adds yet another twist. Not only may there be no need for complex symbolic processing on representations of world models, but the action generation may not be performed centrally at all.

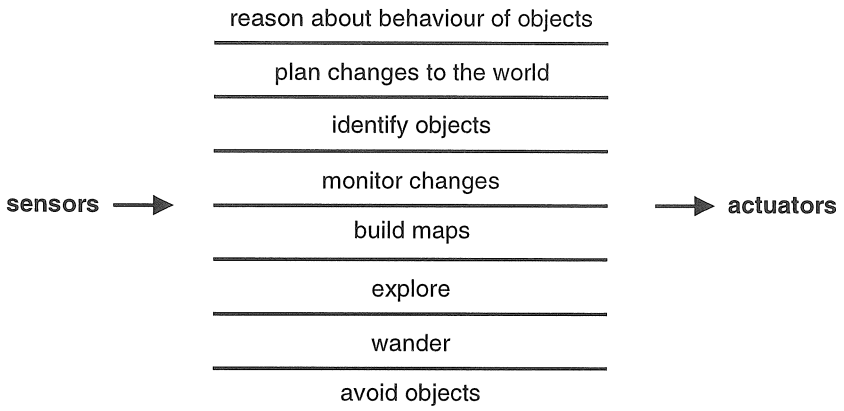


Figure 2.3: Behaviour-based decomposition of a mobile robot control system, based on subsumption architecture [Brooks, 1986].

Brooks [Brooks, 1986] has proposed a new approach for robot motion planning called *subsumption* architecture, that has been applied to control a mobile robot. In this approach reactiveness of responses is combined with distribution of action generation. This architecture is derived from the organisation of biological systems, and is different from conventional architecture to control robots. In the conventional architecture (see fig. 2.1), the information processing is decomposed into a chain of processing modules proceeding from sensing to action. In the subsumption architecture, decomposition is done in terms of behaviour-generating modules, illustrated by fig. 2.3, each of which connects sensing to action. Layers are added incrementally, and newer layers may depend on earlier layers, but do not call them as explicit subroutines.

Another idea has come from Connell [Connell, 1990], who worked on navigation of mobile robot. The robot has no central world models, but a set of distributed, local partial models are also coordinated in a subsumption architecture to achieve the robot's goals in the physical world.

2.4 Learning the Lesson of Biological Systems

How can the nature of non-representational, reactive and distributed approaches be reproduced on a computer and be implemented on a robot ? To find the optimal candidate solution, it is worth looking at how living biological organisms manage the connection of their motor action with sensory perception. It is obvious that such biological organisms' capability is so outstanding and that makes these organisms so superior to our present technical solutions.

The following discussion may bring a new perspective to handle the sensory-motor coordination.

2.4.1 The Escape System of the Cockroach

Almost everyone in the American and European countries is familiar with the cockroach, *Periplaneta americana* , shown in fig. 2.4. This insect has an excellent sensory-motor system to escape the strike of its natural predators.

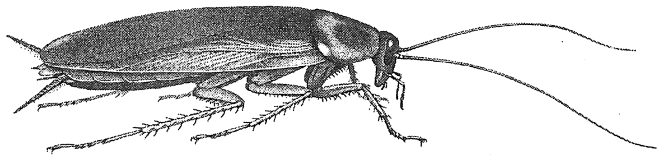


Figure 2.4: *Periplaneta americana*, the cockroach is a native of tropical Africa, but its habits has expanded to include most of the tropical, subtropical and temperate regions of the world [Camhi, 1980].

A laboratorial observation [Camhi, 1980] illustrated in fig. 2.5 shows that a cockroach can flee from a source of wind – the stimulus it uses to detect its predators – in an appropriate direction as soon as 11 milliseconds after an air puff reaches its receptors. How are such behavioural acts initiated and controlled by the insect which has a very primitive nervous system ?

From the engineering point of view, the cockroach behaviour can illustrate how such a reactive behaviour can be achieved by a "system" which has almost no cognitive ability.

Escape Stimulation Mechanism

The cockroach nervous system has a straight and short pathway between the incoming stimuli from its receptors to the excitations of its legs. To escape from predators, cockroaches rely on the stimulation by wind. As shown in fig. 2.6, each wind-receptor neuron is connected to the central nervous system,

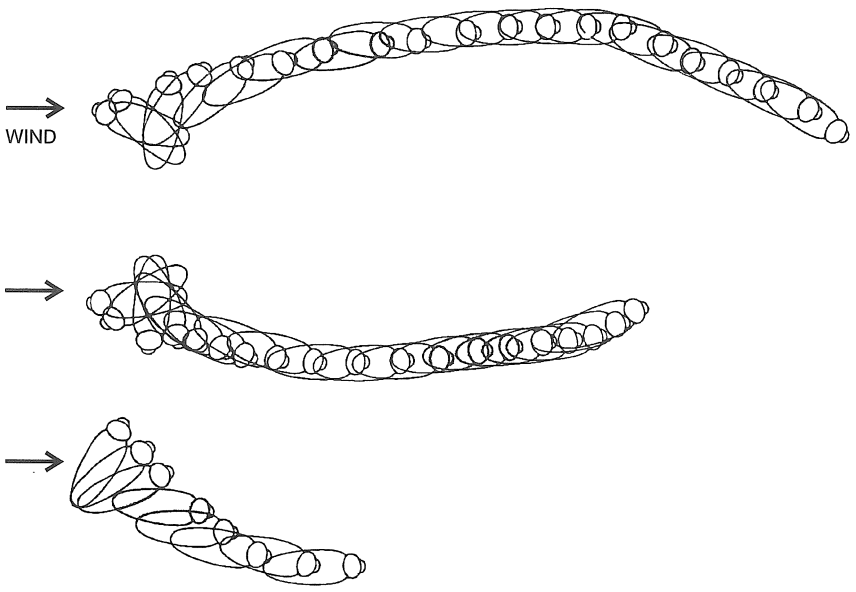


Figure 2.5: Directional responses of the cockroach's escape system in three sample trials, with puffs of air aimed at the insect from different directions. The insect silhouettes at far left, nearest the arrowhead, show the cockroach's positions just before the stimuli. The succeeding silhouettes show the insect's positions at 16 millisecond intervals [Camhi, 1980].

giant interneuron, by its axon. Nerve impulses that originate in wind-receptor neurons are relayed at the terminal ganglia to the giant interneurons. The impulses in turn running up the central nerve cord to the head and pass through the metathoracic ganglia, where the motor neurons controlling the legs are found. The giant interneurons then excite the motor neurons of the cockroach legs.

The mechanism shows that no kind of "brain" organ is involved in this escape circuity. The functional link between sensory neurons and motor neurons is only provided by the giant interneurons.

Initial Running Direction Control

Another interesting behaviour is the cockroach's hard-wired ability to determine the right initial movement direction. Information about the direction of a puff of air is coded in the sensory receptors, which have hairs organised into columns and rows. Particular columns of sensory receptors have a con-

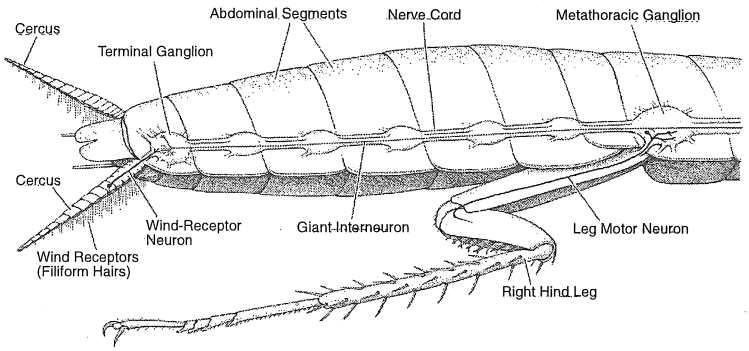


Figure 2.6: Nerve cells involved in the cockroach's escape system [Camhi, 1980].

nection with particular giant interneurons. There are 14 giant interneurons, organised in pairs. Each pair is responsible for the response of each side of the cockroach's body. A group consisting of several pairs of giant interneurons mediate the initial behavioural response and specify the initial direction of turning. It happens thanks to the axons of this group (one for each body side) which conduct action potentials faster than any of the other axons. The turning coordination itself is a kind of "hard-wired preprogrammed". The remaining straight movement is then specified by the rest of the giant interneurons, organised as another group, after being inhibited by the initial turning response.

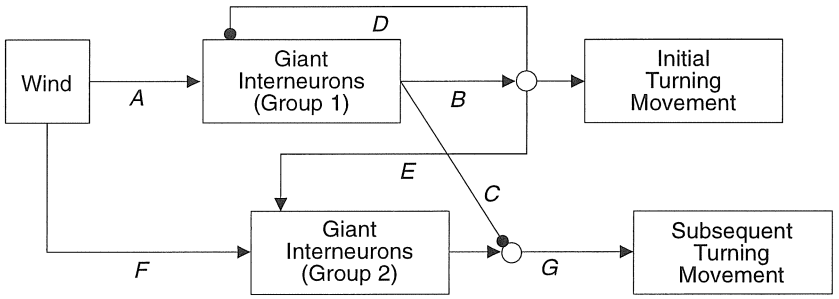


Figure 2.7: Escape behaviour circuit [Camhi, 1980].

Figure 2.7 plots the known interactions among sensory receptors, giant interneurons and other neurons in response to puffs of air. Puffs striking recep-

tor hairs excite group 1 of giant interneurons through pathway *A* about five milliseconds before they excite group 2 of giant interneurons through pathway *F*. Moreover, once activated, the group-1 interneurons inhibit the output of the group-2 interneurons through pathway *C*. At first, group 1 exerts the dominant control over the initial turning movements. Once the turning movements have begun, feedback inhibits the group-1 interneurons through pathway *D* and excites the group-2 interneurons through pathway *E*, so that group 2 may assume dominant control of the behaviour.

What Lesson Does the Cockroach Give Us ?

To perform such a reactive behaviour, which humans most of the time get disturbed by, the cockroach is not supported by a kind of intelligence that is based on a cognition capability. There is no organ that may carry out any explicit "thinking" process between the sensory and the action phase. Instead, its behaviour is more likely produced hard-wiredly, performed by a cooperation of some distributed "simple" processors (neurons) each of which carrying out a "simple operation"; simple in terms of requiring no high cognitive effort.

This gives a view that behavioural actions can be produced through a short and effective information pathway, but involving a number of cooperative and distributed simple processing units. Is it how a real-time behaviour-based system should be built ?

2.4.2 The Human Sensory-Motor Control System

While talking about the human sensory-motor control system, the focus should be put on the human nervous system, which consists of the brain and spinal cord (central nervous system) and all the neural tracts that run from the spinal cord out to muscles, organs, glands, and other tissues (peripheral nervous system).

First of all, keep in mind that what is known currently about the operation of the sensing and moving systems of the human brain and spinal cord, is enough to recognise that they are much more complexly organised than it can be currently explained.

Brain's Anatomical Divisions

The journey into the understanding of the human sensory-motor control system logically should start from the understanding of the functionality of its main *hardware* , the brain. The functional structures of the brain can generally be divided into two categories, *fiber tracts* and *nuclei*. Fiber tracts are simply bundles of axons going from one place to another; they are the cabling and wiring of the brain. The nuclei are groups of neurons. Each nucleus may

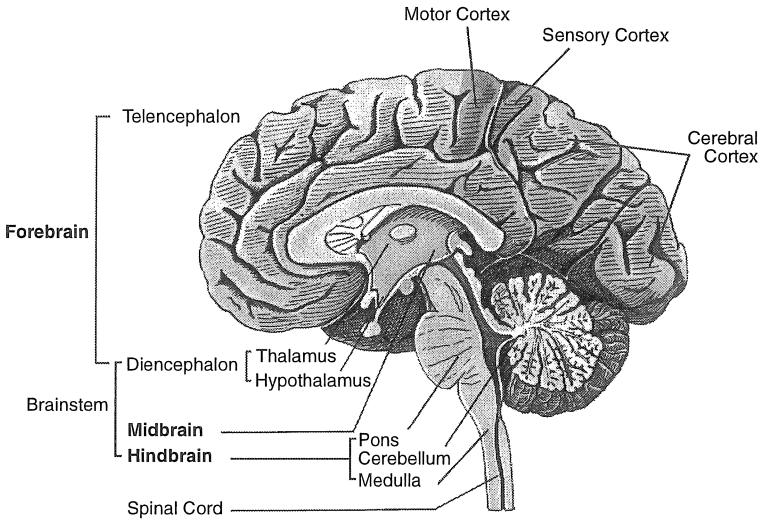


Figure 2.8: The major anatomical divisions and subdivisions of the human brain [Bloom, 1988].

be thought of as analogous to a central processor with a dedicated function. Most of the nuclei are irregular clusters of neurons, but in some, the neurons are arranged in orderly layers which form a folded sheet of cells. In this case the tissue is called a cortex rather than a nucleus. The neurons are near the outer surface of the cortex. The interconnection paths, or the axons, are interior and connect nearby and widely separated brain areas (the cerebral cortex is the pride of human beings since it is better developed in humans than in any other species).

Figure 2.8 shows the major divisions of the brain. The most important items, according to their basic relationship — from bottom to top — are the spinal cord, the medulla, the pons, the cerebellum, the diencephalon (and its two major subdivisions : the thalamus and the hypothalamus), and the cerebral cortex. This bottom-to-top sequence corresponds in a general way to a sequence of increasingly more global levels of control, from the most detailed and specific to the most general and abstract.

The Brain's General Plan

Very simply, the assembly line for sensory-motor processing can be viewed as running in two opposite directions. In the sensory system, information arrives at sensory *receptors* on the periphery and moves up to the cortex. In the

motor system, basic information originates in the motor cortex and ends at the periphery, with the actions of muscle units, or *effectors* . More completely, the sensory-motor system also has its organisational hierarchies and its parallel processing components, and it also relies on the direct sensory-motor maps to work effectively. All of this is applicable whether the movement is a simple one (e.g., involving only one limb), or an elegant and complex one (e.g., involving the whole body activity).

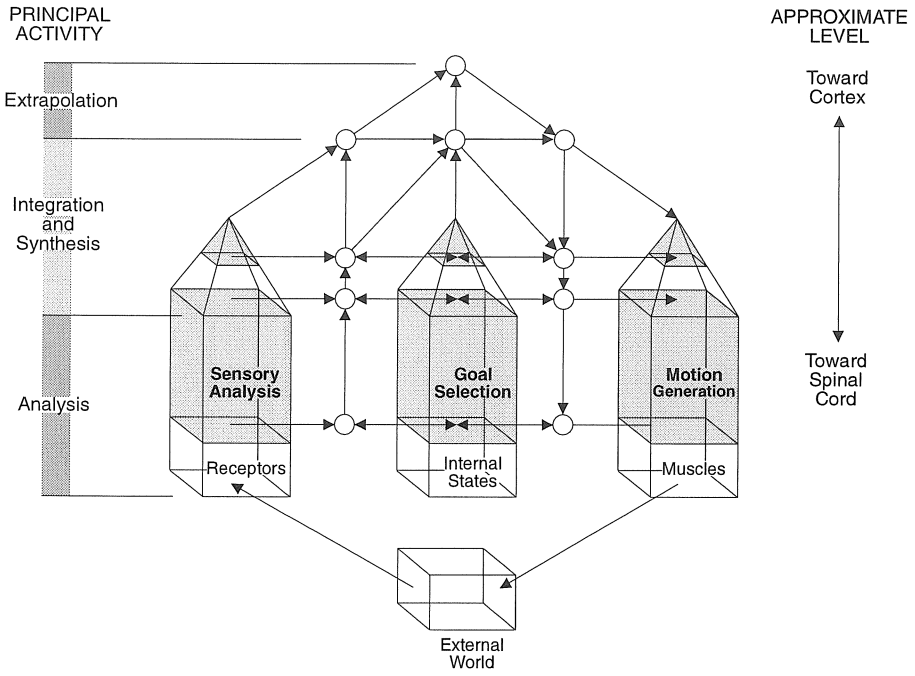


Figure 2.9: A conceptual representation of the basic organisation of human brain's architecture. Square areas indicate data bases in which massive amounts of information are processed simultaneously, in parallel. Circles indicate sites of major interactions between systems [Kent, 1981].

Figure 2.9 shows the basic organisation of the brain's architecture, which is both parallel and hierarchical. The three major functional systems : input, goal selection or "motivation", and output; are shown with their relation to the internal and external worlds, as vertically organised systems. Each of the major systems has a representation at high (extrapolation), low (analysis), and intermediate (integration and synthesis) levels of processing paths.

This bottom-to-top sequence corresponds in a general way to a sequence of increasingly more global levels of control, from most detailed and specific

to most general and abstract. On the input side, the lowest levels gather raw data, which is then progressively abstracted, sorted, and refined at each stage according to general guidelines which may be hard-wired or provided by higher levels. The highest levels then receive abstract symbolic information about the general state of the environment rather than details. Similarly, output function begin at the highest levels, which determine general goals and strategies and transmit these in the form of statements about more limited momentary objectives to lower levels, which in turn send information about desired actions and timing to the lowest levels for execution in detail.

There are three features of such an organisation that are relevant to a system that must deal with the real-world environment and do so in real time. These are : hierarchical decision-making capability, parallel processing of I/O data, and "fail safe" backup function [Kent, 1981].

The first feature is revealed by the organisational levels of the decision-making process. The vast majority of the decisions that have to be made are trivial, and they are handled at lower levels without taking up time, so that the higher levels can use to work on more complex problems. For example, the brain provides a processor for each fiber of each muscle of limbs, for driving each joint of the limbs by continuously computing required forces, based on limb positions or velocities requested by higher levels, while the cortex is doing the planning of the next limb movements.

The second feature, parallel I/O processing, provides the speed needed to deal with a complex world in real time, even in slow biological components. The task is broken down into small parts that can be handled simultaneously by numerous simple processing elements. Thus, the time required for the task is not greater than the time required for an execution needed by one of its components.

The third feature is of high survival value. The redundancy inherent in the brain's basic structure is valuable in this regard. There is an evolutionary order of development of the brain structure, and the major functions have representations at all levels. The brain, in its present state, contains most of the components from its earlier forms. Rather than eliminating the older structures and duplicating their functions, the newer structures simply take control of the older ones and use them as subprocessors.

Components in the Sensory-Motor Path

It may be more useful to identify the functions of some anatomical divisions of the brain which have important representations at its hierarchical levels (shown in fig. 2.8), than to ask where the general function of the brain complexity is performed in the brain.

Intelligent activities occur even from the very early phase of the handling of the sensory clues. Some simple actions in fact proceed entirely at the lowest

level of the central nervous system, the **spinal cord**, where most of the sensory input from the body and most of the output to the muscles passes through. Numerous nuclei within the spinal cord perform many important intelligent functions on both input and output sides, as local I/O processors. The familiar example of this is the "spinal reflex" in the knee-jerk produced by the doctor's rubber hammer.

Further, there is an I/O activity so-called *supra-segmental* level of control, that is carried out by the **medulla** and **pons**. This control level is frequently concerned with directing the activities of the local I/O processors in the cord, to coordinate actions which involve the entire body rather than body members. For example, the pattern of motor activity involved in walking requires the coordination of the whole body to maintain balance as the center of gravity shifts, while the decision to walk and the choice of direction are the province of higher centres.

At the highest level of the hierarchy, the **cerebral cortex** is involved in almost all of the higher functions of the brain structures. Its operation is essentially a vast decoding and encoding that gives analytic and synthetic power to the operation of the other systems, with the capacity for fine distinctions and discriminations on the one hand and large-scale generalisations and syntheses on the other. Two adjacent regions on the cerebral cortex, **motor cortex** and **sensory cortex** , are mainly involved in the sensory-motor operations. The neurons within the sensory cortex are the "projection" of the map of the body's surface. Therefore this cortex is also called somatosensory cortex. The neurons of the motor cortex seem to have a vertical, columnar organisation. It is currently believed that the important function of the cortical motor column is to achieve a specific joint position, not simply to activate specific related muscles. The excitation that drives the motor cortex units arises from units of the sensory cortex at a very late stage in the processing of all forms of sensory information. Highly abstracted information about the position of the body's limbs and the need to initiate movements rapidly is available at this stage, and this information, which includes a full awareness of current joint angles and muscle tension, guides the sensory-motor cortex in activating specific movements.

The whole sensory-motor control mechanism has a delicate monitoring system, known as the *reticular formation*. This mechanism is carried out by the **brainstem**, and has important functions that are analogous to the vectored interrupt system of the computer. It continually monitors the input of all of the sensory systems, more for quantity of activity than for detailed analysis, and controls the degree of activation of various portions of higher centers on this basis. Thus, it can immediately arouse the brain when a novel or intense stimulus is encountered and jump the whole system to a state of attention and analyse the important event.

The entire functioning of the human motor system is also supported by two other important structures that regulate the performance of specific, directed voluntary movements[2] : the **cerebellum** and the **basal ganglia**.

The cerebellum is a subprocessor for some types of motor output. It is viewed as a sophisticated control subsystem which mediates the body's motor activities. The cerebellum does not instigate actions, but takes motor commands from the motor cortex and translates them into smooth, efficient motor programs for the muscles. It is basically involved in the "parallel-to-serial" conversion of output that is not going to be continuously modified by feedback control. It can accept a "parallel signal" which defines an action to be undertaken, modify it to incorporate the current status of many variables of limb position and loading, and convert the instruction to a series of sequenced operations with specified durations.

The basal ganglia[3] or *striatum*, like the cerebellum, is involved in the organisation of motor output. This device generates patterns of movement on the basis of input from analytic cortical areas[4]. It outputs these patterns both to the movement-controlling areas of the cortex and more directly to the lower motor mechanisms. Unlike those generated in the cerebellum, the movements generated in the striatum are under continuous control of the motor cortex, and because the cortex is continually receiving and processing sensory information from the environment, a closed loop system is formed. Besides in the voluntary movements, this system is also very much involved in learned movements.

Sensory-Motor Learning Feature

One of the most important features of the human brain is that it learns. There are at least two types of learning regarding the sensory-motor activities that the brain permits. The first type is learning to associate things that have occurred sequentially several times and which are likely to do so again, and act in anticipation. Thus, if the reflexive response of the nervous system to a painful stimulus applied to the foot is to quickly flex the leg, and if such a painful stimulus is repeatedly preceded by a "neutral" stimulus such as a bell sound, the brain will quickly learn to flex the leg whenever the bell sounds[5]. Assuming the natural reaction to the painful stimulus is of use to the organism, then performing the reaction in response to the antecedent neutral stimulus allows the organism to get a jump on the world and perform more

[2] Voluntary movements are those that one can control when he/she wants to, such as that result when one move his/her limbs, turn the head, or wiggle the tongue.

[3] not shown in fig. 2.8, the name simply refers to the location: at the *base* of the cortex; of certain collections of nerve cells, *ganglia* .

[4] The part of cortex that handles analytical operations.

[5] This is the so-called Pavlovian Conditioned Reflex [Kent, 1981].

efficiently. This learning type does not expand the organism's behavioural repertoire, but just makes it more efficient.

The second type of learning is called "operant conditioning". This type of learning permits humans to expand their behavioural repertoire, based on the quality of the results. Simply it uses the principle that behaviours immediately preceding a reward are increased in future probability of occurrence. "Reward" here refers either to some pleasing event occurring or to some unpleasant state being terminated. Thus, behaviours that lead to good results will tend to recur.

In addition to the kind of learning types described above, which is based on repeated trials and which slowly alters behaviour, the brain also has the ability to store "memories" of facts and events. This is the function humans most commonly mean when they refer to "memory" and recall the same actions they have ever experienced when encountering particular perceptual stimuli.

What Can be Learned from the Human System ?

It is obvious that the way of how humans handle their sensory-motor operations is not the way of which most robotic approaches use to do. Moreover, it is too absurd to try making the artificial version of the complete human sensory-motor system. Yet, the understanding of the human system may inspire new thoughts and "wisdoms" as the foundation for new approaches.

The first thing to be pointed out is that the hardware platform supporting the human sensory-motor capability — the human nervous system — is a structure, whose processing features are distributed, parallel and hierarchical in nature. It is constructed by a collection of cooperative and distributed "simple" processors — neurons — at different control levels and with different processing capabilities. This property, anyhow, gives a new perspective to build the intelligent sensory-motor systems, either through hardware or software implementations. In this way, ANNs become a potential technique to be applied.

Furthermore, human sensory-motor capabilities are generally based on human behavioural developments. Human brains rely heavily on learning to produce most of their behavioural patterns, while the less developed brains such as those of insects, fishes, amphibians and reptiles, which have limited learning capabilities, rely generally on hard-wired inflexible behavioural patterns that are available at birth, and merely as abilities to fend for themselves. The advantage of learned, flexible behaviour is that it can easily adapt to an environment which differs from that in which the species evolved. On the other hand, advanced mammals, particularly humans, are characterised by heavy reliance on learned behaviour, which results in an enormous

behavioural flexibility and adaptiveness to the internal and environmental changes.

2.5 Conclusions

In the first part, this chapter describes approaches which are considered as the classical ways to handle sensory-motor coordination in man-made systems, like e.g. robots : the **built-in-model based approach** and the **traditional AI approach**. They are characterised by a single-processor-based computational technique, which rather modularises the perception, world modelling, planning and action execution. The main common feature of those approaches is their need for accurate representation of the robot and its world. In the case that both can be represented well in advance, these approaches are able to demonstrate their reliability.

The real world, most of the time, is not as simple as what we can predict. The dynamics of the world and the simplifications made in representing them, invite some poor robot performance. To fulfil the need for building systems which can integrate perceptual inputs smoothly with motor responses in the case of lack of world representation, it is worthwhile to look at inspirations provided by biological systems.

Living biological systems have their own way to cope with the world. Many lessons can be learned and adopted from them. The main feature of their natural control systems is that it does not base on a "single processor" computation fashion, but is rather constructed by a collection of cooperative and distributed "simple" processors – neurons – at different control levels, working parallelly. Two examples are presented, which are the primitive hard-wired behaviour of the cockroach and the most sophisticated living behaviour of the human being, which involves high cognitive processes. In all cases, most of the distributed simple processors involve a continuous adaptation, a process performed in terms of learning.

Although there is still much to explore towards the understanding of biological sensory-motor systems, principally it was shown already that the fundamental difference between the current computational approach and the biological evidence in sensory-motor control has been understood. The existing computer architectures were not designed based on the same philosophy as the biological processing system. That fact leads the approaches leading to the present robot sensory-motor systems to have a strong influence from the current computer architecture, characterised by computations on discrete symbolic representations.

So, what is the answer to the question "neuronal or symbolic" ? Particularly speaking in terms of sensory-motor coordination, involving a tight mapping between sensory stimuli and the motor responses, cooperative, dis-

tributed, and parallel processing has to be considered carefully. Above all, the sensory-motor problem is not meant as merely a problem of dealing with perceptual and motor phenomena, but the framework should cover the complex cognitive phenomena as well.

The fact now is that the currently most implemented symbolic approach, is well developed thanks to the history of hardware evolution. The existing computer architectures exhibit to have been fertilised the symbolicism. So again, in encountering the question "neuronal or symbolic", it is wiser to stand at a view that rather by confronting one or the other, disagreements about computations on discrete symbolic representations should turn out to be arguments for computational techniques that support some forms of parallel and distributed processing.

Chapter 3

Learning and Neural Networks in Robot Control

You can build a mind from many little parts, each mindless by itself.
In this scheme each mind is made of many smaller processes. Each process
by itself can only do some simple thing that needs no mind or thought at all.
Yet when we join these processes in societies – in certain
very special ways – this leads to true intelligence.

MARVIN MINSKY

3.1 Introduction

Imagine that someone once spent a long time trying to program a mobile robot to use ultrasonic sensors to navigate along a corridor. He had a physical specification of the environment (it was the corridor he was standing in) and fairly accurate manufacturers' specifications for the sensors and effectors of the robot. Theoretically, he had enough knowledge to write a correct program. However, the specifications of the abilities of the robot and of the properties of the environment were impossible for him to translate directly into a working program. So, he worked in the following debugging cycle :

- Write a program for the robot;
- Run the program on the robot and watch it drive along the corridor;

- Analyse the behaviour of the robot and see where the program was wrong;
- Fix the problem in the program;
- Run the program on the robot and watch it drive along the corridor (again, and this time it is to find another program mistake), and so on.

The result of this cycle was that he learned a good deal about the nature of the interaction between the robot's sensors and the physical environment. Using this information, he *learned* about the environment and *adapted* the robot's behaviour so that it would perform its task correctly. A much more efficient strategy would have been needed for him to design a behaviour for the robot that would, *itself*, adapt to the environment it was in. And therefore, he had to make the robot able to learn to "live" in its environment.

3.1.1 What is Learning ?

Many definitions of learning have been made based on different perspectives and disciplines. In the context of robot learning, learning can be defined as the improvement of a system's behaviour by making it more appropriate for the environment in which the system is embedded [Kaelbling, 1993].

Speaking more generally, computational learning aims to produce an implementation of a mapping between two sets of objects. This is called *target mapping*. [Thornton, 1992] refers a basic model of computational learning to a process in which a *learner* produces a representation of a *target* mapping working from training information derived from some *environment*. This model is illustrated in fig. 3.1.

The process which actually carries out the learning is called the *learner*. The learner is provided with information about the target mapping. When the learning is presented with explicit input/output examples, then the learning process is said to be *supervised* and to carry out *learning from examples*. The process is called supervised because it is reminiscent of the situation in which a teacher (or 'supervisor') helps a learning agent to learn a new concept by providing explicit examples of that concept.

In contrast to supervised learning, there is *unsupervised learning*. In this learning type, the learner does not receive explicit examples of desired input/output association. Rather it has, embodied within it, some more general notion about the way in which inputs should be mapped on to outputs. Very often, this notion is simply the idea that inputs which are similar to one another should be grouped together and associated with the same output. Thus unsupervised learning is often used where the desired goal is to discover how some data divide up into similarity groups.

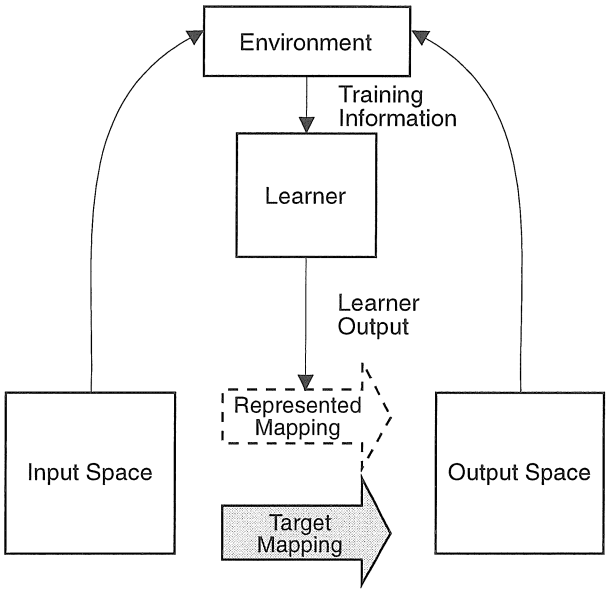


Figure 3.1: A basic model of computational learning [Thornton, 1992].

The term *learner* is sometimes referred to as the learning *agent*. As described in [Steels, 1995], an agent need not necessarily be physically instantiated, but it need to be a system (i.e. a set of elements which have a particular relation among themselves and with the environment). So, an agent can be a computer program (a software agent) as well as physically embodied systems, such as robotic agents or animals.

3.1.2 What to Learn ?

Having decided that the system must learn about its environment, it in turn must be decided exactly *what* the system is to learn. In the end, the system must have some sort of operational mapping from perceptual states to effector actions. One approach is to learn this mapping directly from experience in the world. In this case, the action mapping is tailored for a particular task, so that if the task changes, an entirely new mapping must be learned.

Another approach would be to learn a more general model of the world. If the system could learn a description of the dynamics of the external environment, it would have enough information to satisfy any task within its physical abilities. In order to take action, however, it would need to have a description

of its task and to use that description, together with the learned world model, to decide what to do.

3.1.3 What to Learn from ?

Whether the system is learning the actions map directly or is learning a model of world dynamics as an intermediate stage, it must observe and glean information from the experience it has in the world. In addition to learning how the world works, the system must learn what its task is. It might somehow be possible to program this information indirectly, but in general, that will defeat the purpose of the learning system. There are two considered specifications that will cause the agent to learn to carry out a particular task in a particular environment.

One method is to provide a *teacher*, or other system, to supervise the learning process. The learning system then can be built with the fixed goal of growing to behave in the same way as the teacher, generating the same mapping from perceptual situations to effector actions, even when the teacher is no longer present to be observed.

Another method is to provide a *reinforcement signal*. This is essentially a mapping from each state of the environment into a scalar value encoding how desirable that state is for the agent. The system's goal, in this case, would be to take actions that maximise the reinforcement values received, possible over a long term.

The problem of learning to act is to discover a mapping from perceptual states to actions. Thus, it can be cast as a function learning problem : the agent must learn a mapping from the situations in which it finds itself, represented by streams of input values, to the actions it can perform. In the simplest case, the mapping will be a pure function of the current input value, but in general it can have an internal learning state, allowing the action taken at a particular time to depend on the entire stream of previous input values.

3.2 Learning in Robot Control

3.2.1 Motivation and Goals

From an engineering point of view, it is obvious that some requirements such as the capability of manipulation in unstructured, partially unknown environments and the capability of handling unexpected events, have supported the belief that learning can increase the flexibility of robots as autonomous systems.

Some knowledge contributing to the robot behaviour sometimes is not easy to be represented and downloaded to the robot. To let the robot acquire

and organise the knowledge by itself can be the only effective way to satisfy the desired task. Information necessary to program the robot is, in some situations, simply not readily available. The unknown environment specification, the unknown sensor sensitivity, and the unknown actuator response, can be instants of this situation. In a very dynamic environment, even if there is a complete model of the environment to begin with, this knowledge could quickly become obsolete. Thus it would be beneficial if a robot could constantly update its knowledge of both its internal and external environment.

Figure 3.2 summarises the circumstances and goals underlying the strive toward robot learning implementation.

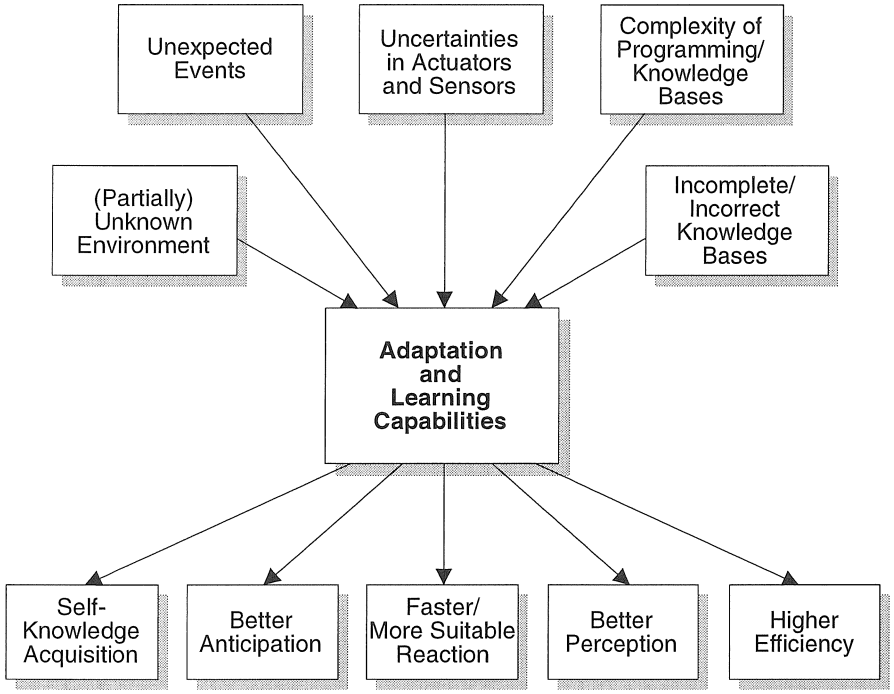


Figure 3.2: The necessity of learning in robotics. Robot adaptation and learning capabilities are required to achieve goals under uncertainty and lack of a priori knowledge [Dillmann, 1994].

When robots are viewed as beings, a relevant analogy and some implications can be drawn from the case of learning in biological systems. In [Brooks and Mataric, 1993], two main benefits of learning in biological systems are identified. First, learning lets the animal adapt to different circum-

stances in the world, giving it a wider range of environmental conditions in which it can operate effectively. Second, learning reduces the amount of genetic material and intermediate structures required for building the complete functioning adult animal. In some circumstances it is simpler to build a small structure capable of constructing a larger one, than to specify the larger structure directly.

The first rationale can be directly applied to robots as well. It can be to adapt the robot to changing external circumstances (e.g. world dynamics), to adapt to changing internal circumstances (e.g. drift in sensors and actuators, unexpected sensor and actuator failure), and to perform new tasks when needed.

The second rationale does not transfer directly to robot control. For instance, many topographical maps within animal brains are built through learning during development instead of being hard-wired genetically[1].

As mentioned in fig. 3.2, implementation of learning robots mainly addresses a number of goals [Dillmann, 1994] :

- Building a robot system that is able to learn to perform better a given task.
- To enable a robot to construct and to modify its knowledge about what is being experienced in the past for possible future use.
- Development of motor and cognitive skills (learning of behaviour).
- Development of error recovery and reactive skills.
- Organisation of knowledge into more effective representations.
- Generation and refinement of real world models.
- Acquisition of declarative knowledge and theories.

3.2.2 Main Types of Learning

In Terms of Knowledge Representation

Presently, computational learning research divides up roughly into two different areas. The first is the area known as *machine learning*. This investigates computational learning using the symbolic approach (see [Carbonell et al., 1984] for further review). The second is the area of ANNs, which investigates computational learning using the non-symbolic approach.

[1] A topographic map is one that preserves locality from sensor space to the neural level (e.g. touch sensors on adjacent patches of skin)

Machine learning focuses on the knowledge acquisition aspect of learning, whereas knowledge acquisition could be considered as belonging to the field of symbolic artificial intelligence research. ANNs as non-symbolic processes come closer to the skill refinement aspect, whereas skill acquisition is inherently non-symbolic in biological systems.

In Terms of Robot Information Level

Specifically in robotics, learning has the purpose of facilitating the actions that the robot takes, of making them more relevant, appropriate, and precise. The level at which the learned information affects action determines the type of learning methods that would be applicable. In particular, based on the type of information that is learned and its effect on the robot's action in the world, [Brooks and Mataric, 1993] divide robot learning approaches into four main categories :

- **Learning numerical functions for calibration or parameter adjustment.** This type of learning optimises operational parameters in an existing behavioural structure.
- **Learning about the world.** This type of learning constructs and alters some internal representation of the world.
- **Learning to coordinate behaviours.** This type of learning uses existing behaviour structures and their effects in the world to change the conditions and sequences in which they are triggered.
- **Learning new behaviours.** This type of learning builds new behavioural structures.

3.2.3 Limitations and Real-World Issues

In dealing with real-world and integrated systems, there are a number of non-trivial real-world issues that must be faced in dealing with learning robot systems. Some of these are [Connell and Mahadevan, 1993] :

- **Nondeterministic actions.** Since the robot has an incomplete model of its environment, actions will not always have similar effects. In this situation planning becomes difficult.
- **Reactivity.** A robot must respond to encountered circumstances in real-time. In many real world situation, delays due to information processing is not acceptable.

- **Limited training time**. To be effective, learning algorithm in a real robot must converge in a relatively short time. Thus, training time available on a real robot is very limited.

Ideally, these issues have to become the design constraints of every learning robot systems.

3.3 Neural Networks in Robot Control

3.3.1 Overview of Existing Applications in Robotics

There are historical parallels between the study of neural networks and robots. Both are biologically inspired in their origin, and both have acquired important engineering applications in their own right. Much work has been done to explore their intersections, their applications and their limitations. Some tried to find the answers to questions like what neural networks can contribute to the improvement of robot control or how faithful a model of biological processing is needed for good integration of perception and action in robotic systems. Much work has demonstrated how neural networks can solve operations in the three levels of robotic processing i.e. task planning, trajectory planning, and trajectory control.

As examples, some applications are briefly mentioned in section 2.3. Many applications deal with learning the *inverse kinematics*, i.e. the mapping relating the world coordinates of a workspace to the joint coordinates of a robot arm. Some applications also involve learning the *inverse dynamics*, i.e. the mapping relates the trajectory of the end-effector to forces and torques exerted at different joints. Other applications are in sensor-based robot control, to learn the sensory-motor mapping, such as visual servoing, fine manipulation (e.g. assembly operation) and mobile robot navigation. Domains such as sensor processing and adaptive control also become fertile fields for neural networks.

Particularly from a robotics point of view, ANNs have the following main advantages over conventional technical solutions :

- High processing speed through massive parallelism.
- Learning and adaptive ability by means of efficient knowledge acquisition.
- Robustness with respect to fabrication defects and with respect to different failures.
- Compact processor models (for hardware design considerations).

3.3.2 Neural Networks Concepts and Definitions

ANNs go by many names such as parallel distributed processors, connectionist models, and neuromorphic systems. Whatever the names, fundamentally, ANN models attempt to achieve good human-like performance via dense interconnection of simple non-linear computational elements operating in parallel. In this respect, the ANN structure is based on the present understanding of biological nervous systems. ANNs model an amount of information simultaneously using massively parallel networks composed of many neurons and connected by links with variable weights. These models have their greatest potential in areas such as speech and image recognition where many hypotheses are pursued in parallel and high computational rates are required.

By definition, as formulated by [Kohonen, 1988], neural networks are "massively parallel interconnected networks of simple, usually adaptive elements and their hierarchical organisations which are intended to interact with the objects of the real world in the same way as biological nervous systems do". From a more hardware-side point of view, [Hecht-Nielsen, 1990] defined a neural network as "a parallel, distributed information processing structure consisting of *processing elements* interconnected via unidirectional signal channels called *connections*". Another point of view of [Zurada, 1992] states "neural networks are physical cellular systems which can acquire, store, and utilise experimental knowledge".

ANN models are specified by a network topology, processing element characteristics and learning rules.The function and performance of ANNs are determined primarily by their pattern of connectivity. Computational elements or nodes used in ANN models are usually characterised by an internal threshold or offset and by their transfer function type, which can be binary, linear or continuous non-linear (as described in **1.6**).

3.3.3 Basic Architectures

The basic components of a neural network are neurons and weighted connections, highlighted in fig. 3.3. A neuron sums all of its weighted inputs and passes the result through an activation function.

Suppose x_i is the neuron input – subscript i indicates the input number – flowing unidirectionally as indicated in the figure by arrows , through a set of adjustable weights, w_i. The neuron output signal, o, is given by the following relationship :

$$o = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{3.1}$$

The function $f(\cdot)$ is referred to as an activation function.

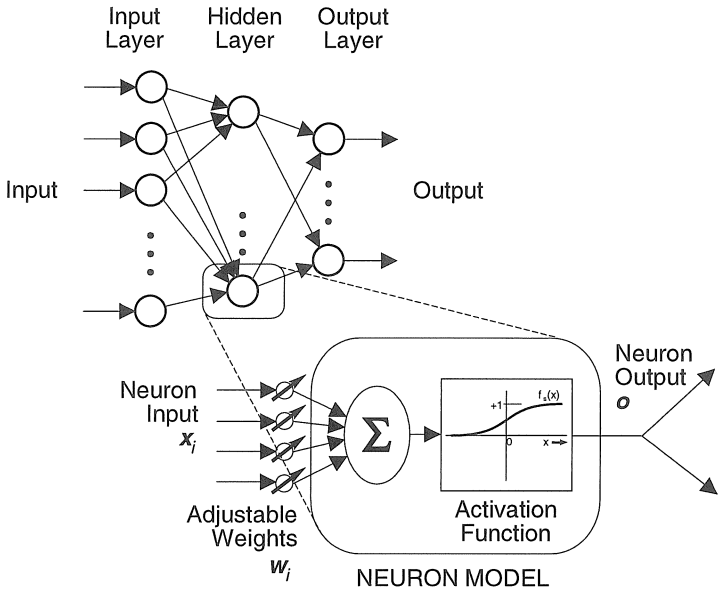


Figure 3.3: A neural network basic architecture.

ANNs may be distinguished on the basis of the directions in which signals flow. Basically, there are two types of neural networks, namely *feedforward networks* and *feedback networks*.

Feedforward Architecture

In a feedforward network, signals propagate in only one direction from an input layer – through intermediate neurons in the hidden layer(s) – to an output layer. The neural network shown in fig. 3.3 is a feedforward network.

Consider a single layer feedforward architecture as shown in fig. 3.4 which consists of m neurons receiving n inputs. The output and input vectors of that layer at instant t are respectively,

$$\mathbf{o} = [o_1 o_2 ... o_m]^t \tag{3.2}$$

$$\mathbf{x} = [x_1 x_2 ... x_n]^t \tag{3.3}$$

The activation value net_i for the i^{th} neuron can be represented as

$$net_i = \sum_{j=1}^{n} w_{ij} x_j \ , \quad \text{for } i = 1, 2, ..., m \tag{3.4}$$

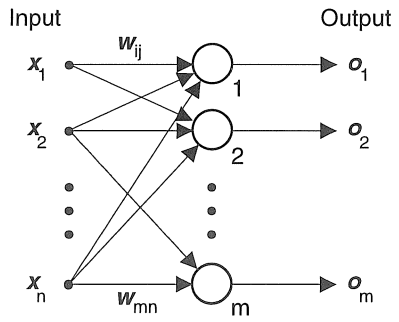


Figure 3.4: Interconnection scheme of a single layer feedforward neural network architecture.

Weight w_{ij} connects the i^{th} neuron with the j^{th} input. The first subscript i denotes the index of the destination neuron and the second subscript j denotes the index of the source neuron.

By involving the activation function, $f(\cdot)$, the processing of neuron inputs is completed. The transformation, performed by each of the m neurons in the network, is a strongly non-linear mapping expressed as

$$o_i = f(net_i)\,, \quad \text{for } i = 1, 2, ..., m \tag{3.5}$$

The mapping of the input space x to the output space o implemented by the whole network can be expressed by introducing the non-linear matrix operator Γ as follows :

$$\mathbf{o} = \mathbf{\Gamma}[\mathbf{W}\mathbf{x}] \tag{3.6}$$

where $\mathbf{W}$ is the *weight matrix* , also called the *connection matrix* , specified as follows :

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \tag{3.7}$$

and

$$\mathbf{\Gamma}[\cdot] = \begin{bmatrix} f(\cdot) & 0 & \cdots & 0 \\ 0 & f(\cdot) & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & f(\cdot) \end{bmatrix} \tag{3.8}$$

The input and output vectors $\mathbf{x}$ and $\mathbf{o}$ are often called *input* and *output patterns*, respectively. Feedforward networks involve no time delay between the input $\mathbf{x}$ and the output $\mathbf{o}$. Thus, equation (3.6) can be rewritten in the explicit form involving time t as

$$\mathbf{o}(t) = \Gamma[\mathbf{W}\mathbf{x}(t)] \tag{3.9}$$

A multilayer feedforward network is created by connecting several layers in cascade. In such a network, the output of a layer is the input to the following layer. The output values of the entire network are compared with the "teacher's" information, which provides the desired output value, to produce an error signal. Feedforward networks usually use this error signal to adapt the network's weights.

Feedback Architecture

In a feedback network, signals also propagate back to the same neuron. Feedback networks can be obtained from the feedforward networks by connecting the neurons' outputs to their inputs, as shown in fig. 3.5.

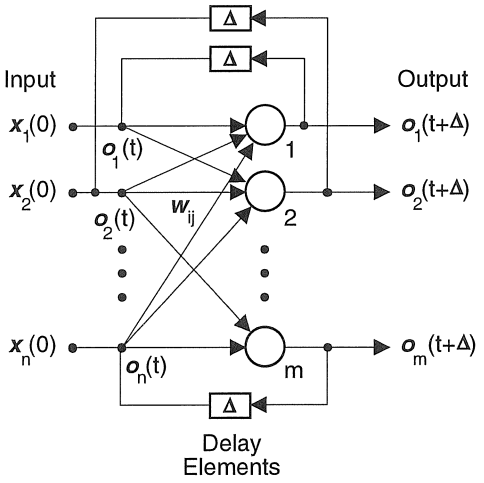


Figure 3.5: Interconnection scheme of a single layer feedback neural network architecture.

The essence of closing the feedback loop is to enable control of output o_i through outputs o_j, for $j = 1, 2, ..., m$. Such control is especially meaningful if the present output, say $\mathbf{o}(t)$, controls the output at the following instant, $\mathbf{o}(t + \Delta)$. The time Δ elapsed between t and $t + \Delta$ is introduced by the delay

elements in the feedback loop. Using the notation introduced for feedforward networks, the mapping of $\mathbf{o}(t)$ into $\mathbf{o}(t+\Delta)$ can now be written as

$$\mathbf{o}(t+\Delta) = \mathbf{\Gamma}[\mathbf{W}\mathbf{o}(t)] \tag{3.10}$$

Note that the input $\mathbf{x}(t)$ is only needed to initialise this network so that $\mathbf{o}(0) = \mathbf{x}(0)$. The input is then removed and the system remains autonomous for $t > 0$. Time t is usually considered as a discrete variable equated to unity, and the time instances are indexed by positive integers, e.g. Δ, 2Δ, 3Δ. For this discrete-time network system, equation (3.10) can be converted to the form :

$$\mathbf{o}^{k+1} = \mathbf{\Gamma}[\mathbf{W}\mathbf{o}^{k}], \quad \text{for } k = 1, 2, ... \tag{3.11}$$

where k is the instant number. This feedback network is also called *recurrent network* since its response at the $k+1^{th}$ instant depends on the entire history of the network, starting at $k = 0$. The network begins its state transitions at starting instant 0, and goes through the state transitions until it possibly finds an equilibrium state, often called an *attractor*.

In some cases, feedback networks also apply *intra-layer* connections or lateral connections, which are connections between neurons in the same layers.

3.4 Neural Processing Levels

Two levels of processing take place at different time scales in the neural learning process. A distinction is made between the *short-term* and *long-term* levels of processing. Long-term processing is what is referred to as *learning*, which is in fact the *adaptation* to modify the connection weights. The short-term processing on the other hand, which has a smaller scope than the long-term one, refers to the dynamics of a single neuron activation.

3.4.1 Short-Term Level of Processing

The *short-term level of processing* consists of the propagation of activity through the network according to the following expression :

$$o_i(t+\delta t) = f\left[\sum_{j \in I_i} w_{ij}(t) x_j(t)\right] \tag{3.12}$$

where o_i is the state (or output) of neuron i, I_i is the set of inputs to this neuron – which can come from the environment, be the outputs of other neurons, or be the feedback states, w_{ij} is the synaptic weight of the connection from neuron j to neuron i, t is the time instant – in this equation time is not restricted to be discrete, δt is the neuron state updating interval, and $f(\cdot)$ takes usually

the form of an activation function. The short-term processing, or dynamics of the neuron activation, has computational interest only in the case of feedback networks.

3.4.2 Long-Term Level of Processing

The *long-term level of processing* is what is conceptualised as learning, and it relies on the application of learning rules that handle the adaptation process of the networks. The majority of the learning rules postulated work by modifying the synaptic weights, according to the following generic expression :

$$w_{ij}(t + k\ \delta t) = g(w_{ij}(t), x_j(t)_{j \in I_i}, x_i(t), \bar{e}(t)) \tag{3.13}$$

where $g(\cdot)$ is a polynominal function, $\bar{e}(t)$ is the learning information supplied at time t, and k is a constant large enough to allow for the propagation of activity through the network before the weight update is carried out.

3.5 Neural Processing Forms

Learning in a neural network is a computational process to adapt the connection weight matrix in such a way in order to achieve the desired network input-output mapping. As a result of learning, data are stored in a network, in the form of connection weight values. Neural network learning is a direct process, which means that each learning step can be captured in a distinct cause-effect relationship. This process is different from learning in human beings or animals, which is inferred; thus its learning step cannot be seen directly unless by observing changes in performance.

After a learning process, a network input-output mapping can be accomplished. The network computation of o for a given x is known as *recall*. Recall is the proper processing phase for a neural network, and its objective is to retrieve the internally stored information. Recall corresponds to the decoding of the stored content which may have been encoded in a network previously.

Another form of neural computational process is *auto-association*. Assume that a set of patterns is stored in the network. If the network is presented with a pattern similar to a member of the stored set, the auto-association process may associate the input with the closest stored pattern.

A related computational process to auto-association is *hetero-association*. Hetero-association processing stores the associations between pairs of patterns. If a pattern presented at the input, hetero-association presents at the output the second member of the input pattern pair.

Classification is another form of neural computation. Classification can be understood as a special case of hetero-association. In classification processing,

the association is between the input pattern and the second member of the hetero-associative pair, which is supposed to indicate the input's class number. If the network's desired response is the class number but the input pattern does not exactly correspond to any of the patterns in the set, the processing is called *recognition*. When a class membership for one of the patterns in the set is recalled, recognition becomes identical to classification.

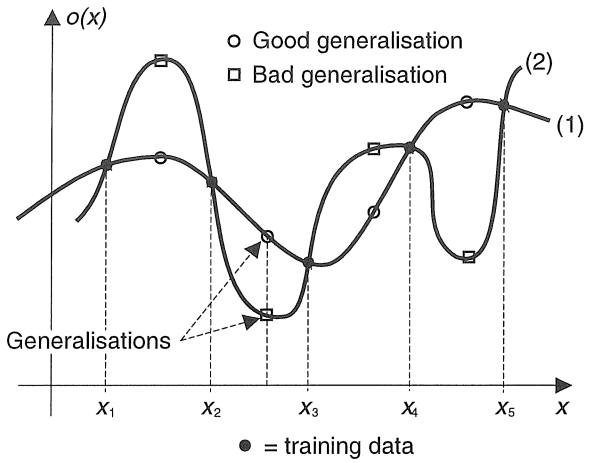


Figure 3.6: Example of neural network generalisation ability [Zurada, 1992].

One of the distinct strengths of neural networks is their ability to generalise. The network is said to generalise well when it sensibly interpolates input patterns that are new to the network. Assume that a network has been trained using the data x_1 through x_5, as shown in fig. 3.6. The figure illustrates bad and good generalisation examples at points that are new and are between the training points. Neural networks provide, in many cases, input-output mappings with good generalisation capability.

3.6 Neural Learning Approaches

Neural learning approaches differ in the type of training information they use. In one extreme one finds *supervised learning approaches* , which require complete target information – in the form of input-output pairs – and whose goal is to build a mapping from inputs to outputs that generalises adequately. *Unsupervised learning approaches* are placed at the opposite extreme, since they use no information at all about the problem and their goal is to carry out feature discovery or clustering, i.e. to build a mapping from inputs to

statistically salient features of the input population that permit establishing clusters of input patterns with similar features. Somewhere in between both extremes lie *reinforcement learning approaches* , which make use only of a single reward/penalty signal – i.e. an assessment of how good is an output to a given input – and whose goal is to build a mapping that maximises reward.

3.6.1 Supervised Learning Approaches

In supervised learning, or learning with a "teacher", at each instant of time when the input is applied, there is always an information about the desired response of the system, provided by the teacher.

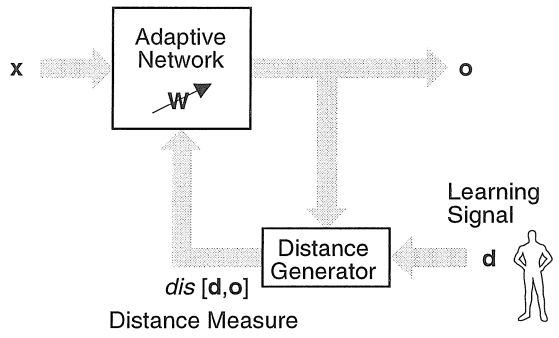


Figure 3.7: Block diagram of supervised learning approaches.

As illustrated in fig. 3.7, the distance dis $[\mathrm{d}, \mathrm{o}]$ between the actual response o and the desired response d serves as an error measure and is used to correct network parameters externally. The teacher may implement input patterns or situations with known responses – in the form of input-output pairs – to adapt the network's weight matrix $\mathbf{W}$ so that the error decreases. A set of input and output patterns called *training set* is required for this learning approach.

Error minimisation rules are used to reduce the error, principally by estimating the negative error gradient direction. In many situations, the inputs, outputs and computed gradient are deterministic, however, the minimisation of error proceeds over all its random realisations. As a result, most supervised learning algorithms reduce to stochastic minimisation of error in the multi-dimensional weight space.

In the following, as examples, two fundamental deterministic error minimisation techniques, i.e. **delta learning rule** and **LMS** *(least mean square)* **learning rule**, are presented.

Delta Learning Rule

The delta rule was introduced for neural network training by McClelland and Rumelhart (1986). The delta learning rule can be generalised for multilayer networks. It is only valid for continuous activation functions. The learning signal for this rule is called *delta* and is defined as

$$\delta \triangleq [d_i - o_i] f'(net_i) \tag{3.14}$$

where $f'(net_i)$ is the derivative of the activation function $f(net_i)$ with respect to its input. The weight adjustment rule is expressed by the equation :

$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(net_i)\mathbf{x} \tag{3.15}$$

where c is an arbitrary positive constant, which can also be represented as learning rate η that determines the speed of learning.

LMS Learning Rule

The LMS learning rule, or also known as Widrow-Hoff learning rule (introduced by Widrow in 1962) is expressed by the equation :

$$\Delta \mathbf{w}_i = c(d_i - o_i)\mathbf{x} \tag{3.16}$$

This learning rule can be considered a special case of the delta learning rule, by assuming $f'(net_i) = 1$, or the activation function is simply the identity function $f(net_i) = net$.

So far, the error minimisation rules are applicable for a single layer neural network. An extension is needed for the application to the multilayer feedforward networks. A most commonly used learning algorithm for the multilayer feedforward networks is called **back-propagation**, introduced by Rumelhart (1986). As its name indicates, it proceeds by propagating error signals from the output layer back to the input layer through all hidden layers, so that appropriate adaptations can be applied to all connection weights.

3.6.2 Unsupervised Learning Approaches

In unsupervised learning, the information about the desired response is not available. In this case, the improvement of the network behaviour cannot be based on explicit error information. To carry out the network's adaptation, unsupervised learning algorithms observe responses to inputs, since no information is available as to correctness or incorrectness of those responses. Inputs usually provide marginal or no knowledge about the outputs, and this

kind of learning corresponds to minimal a priori information available. The observation of the responses to inputs is accomplished by carrying out an internal network processing loop, as illustrated in fig. 3.8.

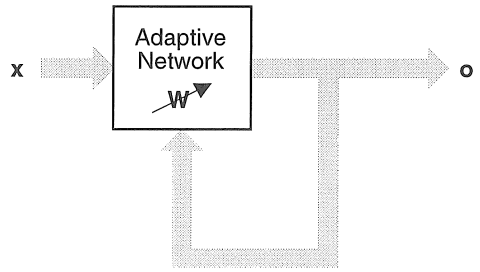


Figure 3.8: Block diagram of unsupervised learning approaches.

The unsupervised learning approaches are often used to perform clustering for the unsupervised classification of objects without providing information about the actual classes. Unsupervised learning algorithms use patterns that are typically redundant raw data having no labels regarding their class membership. In this mode of learning, the network must discover for itself any possibly existing patterns, regularities, separating properties, and so on. Some information about the number of clusters, or similarity versus dissimilarity of patterns, can be helpful for this mode of learning. While discovering the input regularities, the network carries out change of its parameters, which is called *self-organisation.*

These learning approaches usually rely on some variations of the classical **Hebbian learning rule** whose expression in terms of the generic neuron model is

$$w_{ij}(t+1) = w_{ij}(t) + cx_j(t)x_i(t) \tag{3.17}$$

where c is a positive constant that determines the speed of learning.

The most extensively studied application of the Hebbian rule is the implementation of *associative memories.* Associative memories belong to a class of neural networks that learn according to a certain recording algorithm. This class of neural networks performs associations for pattern retrieval and restoration. An associative memory can store a set of patterns as memories and the stored memory can be recalled through association of the information memorised. Associative memory usually enables a parallel search within stored data. It is believed that biological memory operates according to associative memory principles.

The Hebbian rule has also been incorporated into *competitive learning* models that are able to accomplish feature discovery tasks. As shown in fig 3.9,

these models consist of a set of hierarchically layered neurons, each neuron receiving excitatory input from the layer immediately below. Furthermore, the neurons in each layer are grouped into disjoint clusters, each neuron in a cluster inhibiting all other neurons within the cluster.

The name of "competitive learning" comes from the fact that the neurons within a cluster "compete" with one another to respond to the pattern appearing on the layer below. The more strongly any particular neuron responds, the more it shuts down the other members of its cluster.

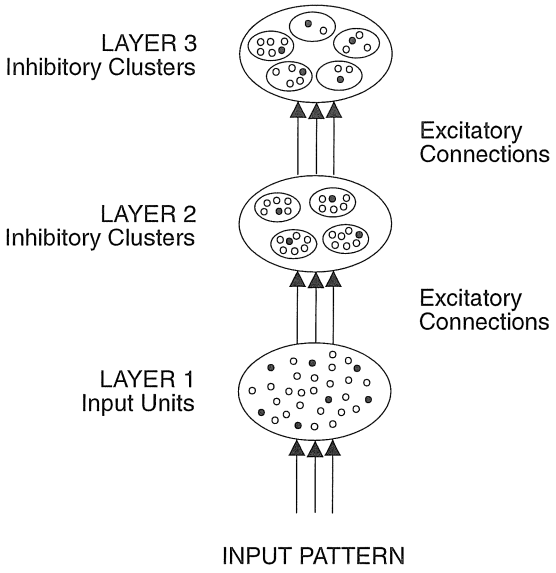


Figure 3.9: Generic form of competitive learning models. Active neurons are represented by filled dots, inactive ones by open dots. The neurons within a cluster inhibit one another in such a way that only one neuron per cluster may be active.

If the stimulus patterns naturally fall into classes, the system will find exactly these classes and the attained classification will be very stable. However, when presented with an arbitrary input environment, competitive learning models can become very unstable and the need appears of stabilising their response through the use of specialised mechanisms, leading to *adaptive resonance* models [Grossberg, 1987]. These are recurrent modular networks able to perform a new cluster whenever they are presented with an input pattern that is very different from the patterns previously seen.

Following the same line of competitive learning, it has been proposed to use *self-organising feature maps* to construct mappings that preserve topography (i.e. neurons that are spatially close in the network are maximally acti-

vated by input vectors close according to the Euclidean metrics). Essentially, this is realised through two-layer networks with intralayer lateral and interlayer connections. Work employing self-organising feature map networks in robotics have been carried out by [Ritter et al., 1992] and [Lee et al., 1992] in the domain of hand-eye coordination.

3.6.3 Reinforcement Learning Approaches

In many cases, an adaptive system is designed to operate in a tight, closed-loop interaction with its environment, where some sense of behavioural adaptation to variations in the environment is required. In some situations, there is less detailed information available as feedback from the environment, which is in the extreme case even only an *evaluative*, not *instructive*, feedback signal obtained from the environment saying whether the output is *right* or *wrong*. It is assumed earlier that supervised learning approaches require an exact desired response of each input pattern, so they are not applicable for this situation.

Reinforcement learning approaches come as a possible solution to this kind of situation. Informally, reinforcement learning is defined as learning by trial and error from performance feedback, i.e. from feedback that evaluates the behaviour generated by the learning agent, but does not indicate the correct behaviour. The feedback signal from the environment is called the *reward*. Unlike the sensory information, which may be a large feature vector, or the action, which may have many components, the reward is a single real-valued scalar. The goal of adaptation is the maximisation of the cumulative reward received over time. Reinforcement learning is sometimes called **learning with a critic** as opposed to learning with a "teacher", since a reward – if it is negative – can be also considered as a critic for the learning agent.

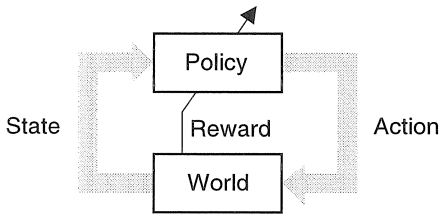


Figure 3.10: Block diagram of policy-only learning architecture.

All reinforcement learning algorithms involve the learning of a mapping from a representation of a situation or *state* to an appropriate *action* (or a probability distribution over actions) for that situation. This mapping is called the *policy*; it specifies what the learning agent will do in each situation at its cur-

rent stage of learning. The simplest reinforcement learner would consist only of a policy and a way of adjusting it based on reward, as shown in fig. 3.10. Such architectures, in which the policy is the only modifiable data structure (and the only structure at all) are usually called *policy-only* architectures. Examples of such learning architectures are presented in [Barto and Sutton, 1981, Barto and Anandan, 1985].

The learning algorithms for the policy have to take into account the fact that a desired action is not directly available. Instead, a form of correlation must be established between the reward received and the actions taken by the learning agent, all with respect to the sensory input. Actions correlated with high reward have their probability of being repeated increased, while those correlated with low reward have their probability decreased.

Policy-only architectures really only work well when it is clear a priori what constitutes a high reward and what constitutes a low one [Sutton, 1991] – for example, if all high rewards are positive and all low rewards are negative. Often, rewards are not distributed around a baseline of zero, but around some other, unknown value. Worse yet, the baseline may change from state to state. A low reward value in one state may be the highest attainable in other. To handle such variations, a baseline value must be dynamic.

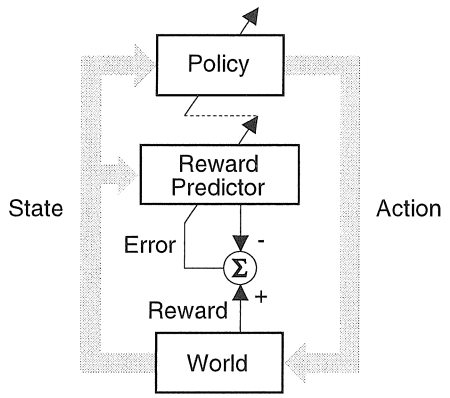


Figure 3.11: Block diagram of reinforcement-comparison learning architecture.

In [Barto et al., 1981, Williams and Peng, 1989], algorithms are explored in which the baseline value is learned as a function of the state, and the actual reward is then compared with the current state's baseline. This is what is done in *reinforcement-comparison* architectures, depicted in fig. 3.11. As a baseline, these architectures usually use a *prediction* of the reward. The prediction error – the difference between predicted and actual reward – is used both as

an enhanced, zero-balanced reward signal for adjusting the policy and as an ordinary error for learning the reward predictions.

Reinforcement comparison architectures are effective at optimising immediate rewards, but not at optimising total reward in the long run. The problem is that actions have two kinds of consequences – they affect the next reward and they affect the next state, but reinforcement-comparison architectures only take the first of these into account. This leads to a failure if suppose an action produces high immediate reward but deposits the environment in a state from which only low reward can be obtained in the future. In order to optimise long-term reward, these delayed effects of action must be taken into account.

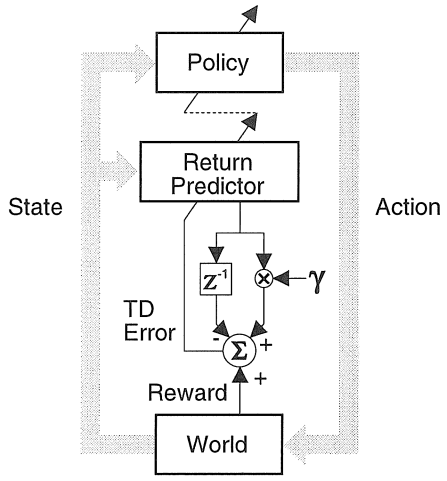


Figure 3.12: Block diagram of adaptive heuristic critic learning architecture. The symbol $\mathbf{z^{-1}}$ indicates a one step delay and the symbol $\otimes$ indicates multiplication.

An architecture of a so-called adaptive heuristic critic (AHC) came up with a design which takes such delayed effects into account – as depicted in fig. 3.12 [Barto et al., 1983, Anderson, 1987, Barto et al., 1991]. The predictor of immediate reward has been replaced with a predictor of *return*, a measure of long-term cumulative reward. For any state x, the return is formally defined as the expected value of the sum of all future rewards, discounted by their delay, given that the system starts in x :

$$return(x) = E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1}\,|x_o = x\right\} \tag{3.18}$$

where r_{t+1} is the future reward and γ, $0 \leq \gamma \leq 1$, is the discount rate determining how fast one's concern for delayed reward falls off with the length of the delay. The "return predictor" box in fig.3.12 comes to predict this return by virtue of the circuit shown below it for calculating a temporal difference error [Sutton, 1988]. In all other respects, the learning algorithm inside this box could be exactly the same as that used in the "reward predictor" box of fig. 3.11.

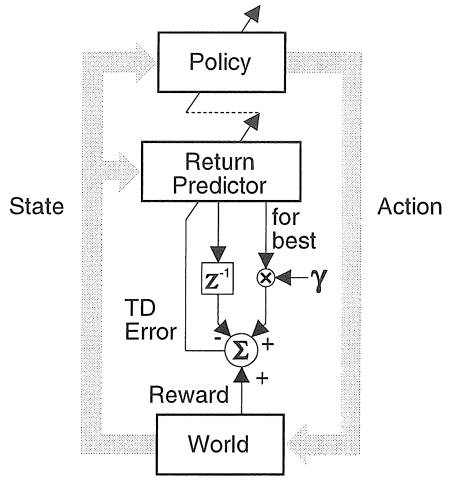


Figure 3.13: Block diagram of Q-learning architecture. The symbol $\mathbf{z^{-1}}$ indicates a one step delay and the symbol $\otimes$ indicates multiplication. The line labeled "for best" is the prediction of return for the best action.

The most recent reinforcement learning architecture is *Q-learning*, proposed by [Watkins and Dayan, 1992]. The primary innovation in this architecture, as shown in fig. 3.13, is that the predicted return is now a function of action as well as state.

Formally, the return for a state x and action a is defined as

$$return(x,a) = E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1}\,|x_o = x, a_o = a\right\} \tag{3.19}$$

Two kinds of return are always predicted for the current state. One is the best predicted return for the state – the predicted return for the action with the highest predicted value. The other is the predicted return for the action actually selected. These two predictions are then combined according to the same circuit as used in the AHC architecture. In Q-learning, the policy could be a separate modifiable data structure as suggested by fig. 3.13, but most

often it is simply a dependent function of the return predictions. For example, the policy may be simply to pick the action that obtains the maximal return prediction.

In general, most of the problems in autonomous robotic systems fall into a category of dealing with the tight close-loop interaction with the environment, to which the development of reinforcement learning approaches are directed.

3.6.4 Incremental versus Batch Learning

The majority of neural networks require training in a supervised or unsupervised learning mode. This mode of training carries out *incremental learning*, which is learning with feedback and is usually performed in steps. Some of the networks, however, can be designed without incremental training. They are designed by *batch learning* rather than by stepwise learning.

Batch learning takes place when the network weights are adjusted in a single training step. In this mode of learning, the complete set of input/output training data is needed to determine weights, and feedback information produced by the network itself is not involved in developing the network. This learning technique is also called *recording*.

3.7 Neural Control Schemes

A number of control schemes can be implemented when involving ANNs in a control system. As intelligent control system, neural controllers can be applied both in adaptive control and learning control [Yabuta and Yamada, 1991]. The adaptive control is a kind of the intelligence which tunes dynamic parameters when the dynamic structure is known. Learning control realises a little higher level control than adaptive control, which can produce an optimal control input to realise desired outputs when the dynamic structure is unknown. Neural networks mostly approach the design of controllers by using the estimated information about the plant. Such information is assumed to be gradually acquired during plant operation.

Various neural control architectures have been proposed in the field of control and system identification. Two basic plant identification techniques that make use of neural networks, namely **forward plant identification** and **plant inverse identification**, will be described. They are applicable in operations which vary in time [Sontag, 1992]. For control, the basic neural control schemes employing back-propagation on multilayer neural networks, namely **general learning** and **specialised learning**, will be described. They are mostly used in the situation where on-line system adaptation is required, both in static and dynamic plants [Psaltis et al., 1987].

3.7.1 Forward Plant versus Plant Inverse Identification

Plant identification can be distinctly helpful in achieving the desired output signal of the plant. Neural network identification techniques offer their capability of learning non-linear mappings to represent the plant behaviour.

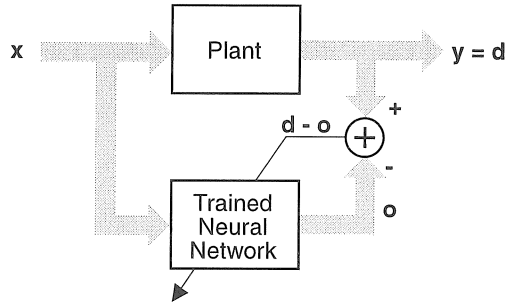


Figure 3.14: Neural network scheme for forward plant identification.

The basic scheme of **forward plant identification** is shown in fig. 3.14. A neural network receives the same input $\mathbf{x}$ as the plant, and the plant's output provides the desired response $\mathbf{d}$ during training. The purpose of the identification is to find the neural network with response $\mathbf{o}$ that matches the response $\mathbf{y}$ of the plant for a given set of inputs $\mathbf{x}$. This implementation typically applies a multilayer feedforward architecture with a back-propagation learning algorithm. During the identification, the norm of the error vector, $\| \mathbf{d} - \mathbf{o} \|$, is minimised through a number of weight adjustments. In this case, the network attempts to model the mapping of plant input to output, with both input and output measured at the same time.

In contrast to forward plant identification, identification of the plant in-

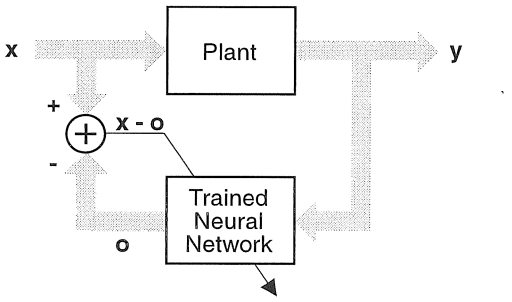


Figure 3.15: Neural network scheme for plant inverse identification.

verse may offer another alternative for designing the control system. In the **plant inverse identification**, the plant output y is used as neural network input, as shown in fig. 3.15. A multilayer feedforward architecture with back-propagation learning algorithm is typically used. The error vector for network training is computed as x – o, where x is the plant input. The norm of the error vector to be minimised through learning is therefore || x – o ||. This scheme of training implements the *mapping of plant inverse*. The trained neural network will act as controller, and the appropriate controlling signal for the plant is directly available at the network's output.

3.7.2 General versus Specialised Learning

In the case of **general learning**, two stages are carried out separately, i.e. the training stage and the control stage. In the training stage, the neural network controller is trained using the plant inverse identification technique. In this stage the plant is used to produce the training set consisting of a set of input/output pairs in a chosen working range, depicted in fig. 3.16.

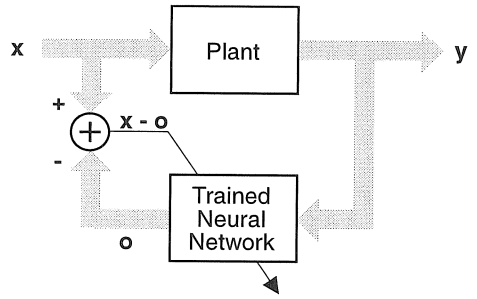


Figure 3.16: Training stage of general neural network learning control scheme.

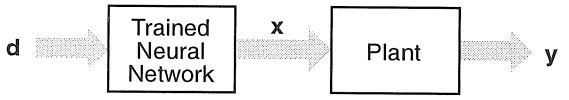


Figure 3.17: Control stage of general neural network learning control scheme.

After the training stage, the neural controller is laid-out as shown in fig. 3.17. In this stage the neural network is able to provide control inputs to reach any

desired targets within the plan working range. This method has several drawbacks. Firstly, learning must be performed off-line. Secondly, the network cannot limit its working range to a desired output range that is actually relevant. Finally, since it learns off-line, the control system is not adaptive. An application example of this control scheme is illustrated in [Kuperstein, 1989] to learn robot hand-eye coordination with a stereo camera.

As an enhancement, an on-line version of the general learning scheme is proposed in [Psaltis et al., 1987], which is called **indirect learning architecture**, as shown in fig. 3.18.

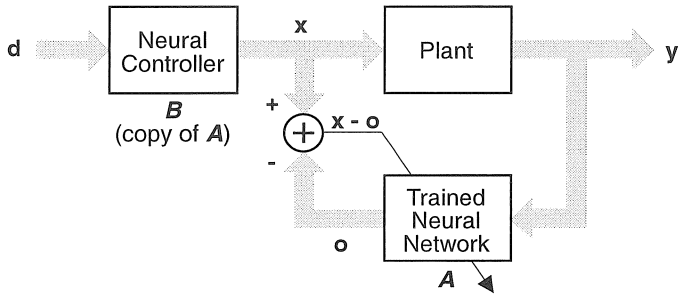


Figure 3.18: Indirect on-line neural network learning control scheme.

Neural controller B is an exact copy of neural network A, which undergoes training. Network A is connected in such a way that it gradually learns to perform as the unknown plant inverse. The input of the controller is $\mathbf{d}$, which is the desired response of the plant. The plant's actual response is $\mathbf{y}$ due to its input $\mathbf{x}$ produced by the neural controller B. The error used for training the neural network is the difference between the output signals of the networks A and B. This error must be reduced until $\mathbf{y}$ can exactly match $\mathbf{d}$.

The advantages of this control scheme is that it is trained on-line, so there is no need of a separate training stage. The neural network also learns continuously and is therefore adaptive. Moreover, the controller's training can be performed in the region of interest of the output vector domain, since the inputs to the neural controller are the desired outputs of the plant. The disadvantage of this scheme is that it becomes useless if the plant inverse is not uniquely defined.

A closely related neural control scheme for control in the region of interest of the output vector domain is called **specialised learning** [Psaltis et al., 1987]. Specialised learning differs from general and indirect learning by the fact that the controller learns no longer from input/output pairs but from a direct evaluation of the network accuracy, with respect to the output of the plant. The network uses the difference between the actual output of the plant and the

desired output to change the weights of the connections. Specialised learning avoids several drawbacks of general learning, that is it no longer needs a separate training stage. It learns continuously, so that it is adaptive.

There are two different specialised learning architectures which are dedicated to static and dynamical plants.

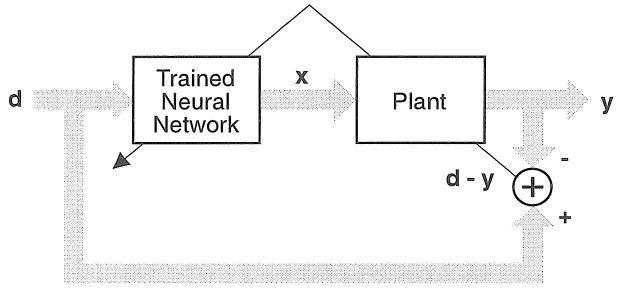


Figure 3.19: Specialised on-line neural network learning control scheme for static plant.

Figure 3.19 shows the specialised learning scheme for static plants. This control scheme possesses all advantages of the indirect learning scheme, but it uses only a single neural network instead of the two required by the indirect control system. The control objective of this scheme is to obtain the desired output d of the plant under the condition that the plant input causing this output is unknown.

Figure 3.20 shows the specialised learning scheme for dynamical plants. In this control scheme, the trained neural network needs to be provided with the sequence of recent output plant vectors in addition to present plant output. This would allow for reconstruction of the current state of the plant based on this most recent output history. The control scheme presented in fig. 3.20 is of order k. In this case, the last k output vectors of the controlled plant are augmenting the present input of the trained neural network. The constant k can be referred to as the order of the dynamical plant [Saerens and Soquet, 1991]. The delay units Δ in the feedback loops denote the time elapsed between the samples of inputs used for training, and that the plant is controlled in real time.

A particular issue in specialised neural network learning control schemes is that there is no direct error signal available for the training of the neural network. The required error signal has to be derived from the evaluation of the error from the plant output, for which in this case a prior knowledge of the plant is required. In [Saerens and Soquet, 1991] a technique called *back-propagation through plant* is proposed to handle that issue.

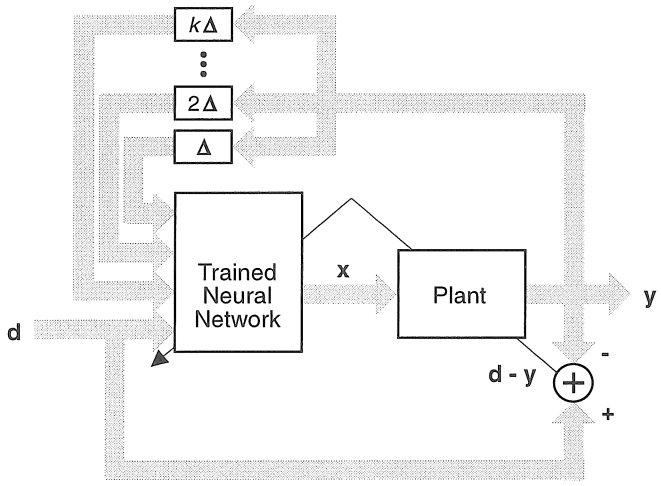


Figure 3.20: Specialised on-line neural network learning control scheme for dynamical plant.

3.8 Limitations and Towards a Hybrid System

The way of how neural networks handle computational problems is very different from the conventional computational techniques. As a summary, table 3.1 provides an overview of the basic aspects of computation for each of the techniques [Zurada, 1992].

The conventional computation has encouraged the philosophy of symbolic computational. In symbolic-based techniques, the knowledge is commonly represented by rules, frames or objects that can be made understandable to humans. On the other hand, neural networks represent knowledge in an implicit manner, namely in the form of weights and activation functions, which are incomprehensible to users. The knowledge structure changes dynamically during the learning phase. Adding or changing a component of the network, mostly means to begin the whole training process nearly from scratch [Chandrasekaran et al., 1988].

The surveyed investigations confirmed that neural network techniques seem to be the most viable solution for the lower level of intelligence, hierarchical control and monitoring systems [Monostori and Barschdorff, 1992, Torras, 1992]. Today's neural network techniques can contribute to the solution of some partial problems, mostly where sensor integration, high processing speed and adaptivity are needed. Neural network procedures seem specially well-suited for dealing in real-time with unknown and dynamic en-

Performance aspect or task	Conventional computation	Neural network computation
Problem solving	Algorithm formulation	Selection of architecture and definition of the set of representative examples / stimuli
Input data	Numerical form	Numerical, but also perceptual representation allowed
Knowledge acquisition	Programming	Training
Knowledge retrieval	Sequential computation	Recall in the form of collective processing
Computation	High precision arithmetic	Low precision, non-linear mapping
Internal data	Internal representation in control of the algorithm	Internal representation in control of input data
Fix or intermediate data storage	ROM, RAM high precision binary memories	Interconnecting weights of typically continuous values

Table 3.1: Comparison of conventional and neural network computation.

vironments. However, they suffer from an extremely time-consuming learning phase and they have no way to encode *a priori* knowledge about the environment to gain efficiency [Torras, 1992]. Since the higher levels of control/monitoring hierarchy require mostly symbolic knowledge representation and processing, the integration of symbolic and neural network methods can be potential. The development of trully intelligent autonomous systems needs further development of these two methods. Significant developments can probably be reached by taking the best of both methods by building hybrid systems. Special emphasis must ge given to the integration of these two knowledge representation and processing techniques, and on the conversion of knowledge between these forms [Chandrasekaran et al., 1988].

3.9 Conclusions

This chapter treats introductory concepts and definitions used in robot learning and artificial neural networks. Since potential applications of learning to robots are varied and diverse, the right and careful selection of a learning paradigm is very important.

The basic concept of neural networks as feedforward and feedback recall systems is provided in this chapter. From a robotic point of view, it leads to the understanding of the kind of processing tasks they can handle in relation with the sensory-motor problems.

According the the kind of learning stimulus received, two classes of approaches can be presented, which have some analogies with the approaches defined in the experimental psychology. The first are learning approaches that are based on training for a stimulus-response sequence, which are referred to by the psychological term of *classical conditioning*. The second are learning approaches that reinforce only the responses of a learning agent, which are referred to by the psychological term of *operand conditioning*. From this conception, the selection of the learning approach must be strongly based on the available stimulus in a learning robot system.

Control system architectures based on neural networks can be used as models of dynamical systems, such as robots, whose outputs are dependent on input history. Neural networks seem to be especially promising for non-linear dynamical system control problems. In this respect, suitable control schemes and more systematic modelling approaches need to be devised for this purpose.

A combination of symbolic learning for high-level tasks (e.g. planning) and neural learning for low-level activities (e.g. control through sensory feedback) would enhance the capabilities of autonomous robots. The key point to achieve such a combination is to devise good interfaces between the representations used in the two paradigms.

Chapter 4

Tactile Perception as a Holistic Process

We must see the matter at once, at one glance, and not by a process of reasoning, at least to a certain degree . . . Mathematicians wish to treat matters of perception mathematically, and make themselves ridiculous . . . the mind . . . does it tacitly, naturally, and without technical rules.

PENSÉES PASCAL

4.1 Introduction

It is easy to take perception for granted. A glance around our world seems to provide us with an immediate "understanding" of our environment : shapes, sizes, and colours of objects and their interrelationships in the 3-dimensional world. This understanding also seems to be valued rather cheaply, the effort is certainly less than solving most other cognitive problems, such as chess or mathematical proofs. Yet, perception, as well as language and thinking, remain the exclusive province of the biological systems and beyond the reach of machine perception by current computers [Hoffman, 1986].

The problems of machine perception are analogous to those of human perception that have been studied for a long time by psychologists [Nevatia, 1982]. But machine perception is currently a long way from the ultimate goal of spec-

ifying the information-processing architecture in sufficient detail to build a perceptive machine. At present, machine perception can only sketch pieces of the puzzle and identify the basic problems that need solution [Hoffman, 1986].

Perception is not simply a processing of sensory input. As illustrated by human perceptual activities, perception is not a reading by a feature extractor, but an active intentional interpretation of sensory clues [de Callataÿ, 1986]. Engineering approaches to machine perception mostly analyse the complex sensory signals into components by a set of independent feature-analysing systems. In this approach, the large sensory input space is transformed to the smaller feature space. Perceptual experience occurs late in the processing sequence after features have been reassembled. In this case, selection of good features is a crucial step in the process since the next stage only sees these features and acts upon them.

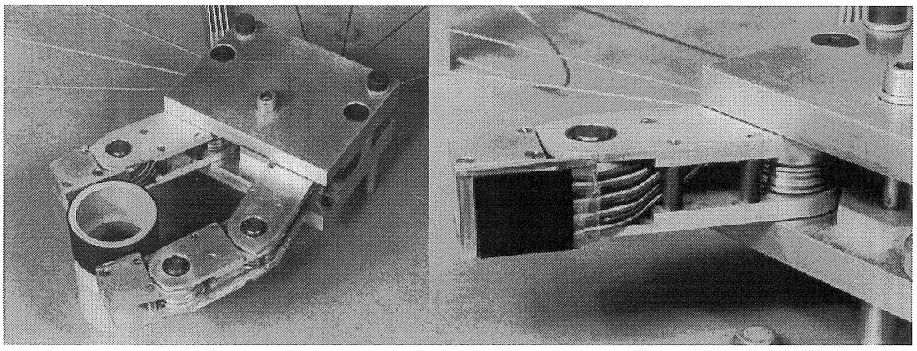
It is argued that perceptual activities in humans do not proceed by combining evidence about constituent features of a sensory pattern but, rather, uses a holistic process [Dreyfus, 1979]. For example, human recognition of a familiar face or voice, is performed with this holistic character. The visual impression a human obtained from his/her sight to another's face is directly mapped to an identity he/she knows and this is not achieved by composing evidence about features, such as eye colour, mouth shape, hair style and so on. The same process also occurs in recognising voice.

Additionally, humans develop their abstraction through learning. Humans start with an initial set of abstractions – that a child starts to develop – to which the learning process adds sufficient new abstractions to solve the task. The perceptual capability that results after learning has a natural interpretation. The learning process can be interpreted as having successfully searched the space for the information processing abstractions which are needed to solve the problem [Chandrasekaran et al., 1988].

This chapter will focus on the contradictory issue between the feature-based approach and the holistic approach in the case of robot tactile perception. Some insights to the issue will be based on experiments. Neural networks are a candidate paradigm to build learning robot perception systems. Although neural networks are considered to be still far from mimicking functions of the human nervous system, it is believed that they are characterised by a tremendous potential, highly parallel operation, and fault tolerant nature that make the neural network solutions are holistic [Chandrasekaran et al., 1988]. In this case, the use of neural networks can be justified by nature's tremendous performance in featureless direct recognition.

An experimental case study in robot touch perception problem is chosen to illustrate the insights. Adding the touch perception in a robot gripper is probably today's most promising technique for object recognition and for controlling a robot hand using a more intelligent manipulation algorithm. In many

cases, robot touch sensing problems are similar to robot visual sensing problems. Perception of touch is another example whose information processing can be explained without composing features.



(a) the two-fingered gripper (b) the tactile sensor surface

Figure 4.1: The K.U. Leuven two-fingered gripper used in the experiment of object classification to illustrate holistic perception (a), and the tactile sensor surface incorporated on the thumb of the gripper to obtain the impression of the objects (b). *Photograph by courtesy of D. Reynaerts.*

A problem of classifying objects gripped by a robot gripper will be investigated. The classification is done based on the tactile images obtained from a tactile sensor surface mounted in a gripper finger as shown in fig. 4.1. Both feature-based and holistic approaches will be implemented on neural networks. Their performances will be evaluated and a comparison to some statistical pattern classifications will also be made. From this experimental study, some evaluative remarks on the holistic nature of machine perception can be drawn.

4.2 Psychology of Perception

Perception invites a multidisciplinary approach. The results of research in psychology as well as insights gained by human introspection have guided much of the research in machine perception, even though the motivation is not necessary to simulate human perception [Nevatia, 1982]. Experimental psychologists start with the behaviour of a perceiving organism and hope that it will allow insights into the underlying representation and processes of perception.

4.2.1 Bottom-Up and Top-Down Analysis

A perceptual system needs to accomplish several abilities for processing the sensory data from its sensory receptor until the desired information is obtained. The processes may be considered to form a hierarchy of abstraction levels. The representation of a sensory impression could benefit from two different kinds of above processes : a *bottom-up* analysis, which takes data present in the input image and constructs a representation of the component objects, and a *top-down* analysis, where *a priori* knowledge – about objects and interrelationships that usually occur together – is available. The human perceptual system appears to use both of these processes in a seamless interaction to arrive at an interpretation [Hoffman, 1986].

Bottom-Up Analysis

In bottom-up knowledge processing, the information flows from one level to the next higher level of abstraction without any influence from the expectations of this higher level. Human ability to perceive unexpected or unfamiliar scenes requires capabilities of bottom-up processing, to the level of meaningful object descriptions. However, extensive communication between the various levels is needed, hence the processing is not strictly bottom-up. In machine perception, this type of analysis is applicable in the case of e.g. identification of new patterns.

Top-Down Analysis

In top-down processing, the processing at a lower level is specifically directed to satisfy an expectation or goal of a higher level. At a higher level, the goal may be to verify if a certain object is present in the scene. Human capability of top-down processing is indicated by the human ability to see a suggested object in a confusing scene – for example, a suggested pattern in the clouds in the sky. In machine perception, this type of analysis is applicable in the case of pattern classification.

4.2.2 Holism as a Concept : an Illustration

Holism, which is used as the underlying concept in this chapter, is actually borrowed from the psychological terminology. This section will investigate the meaning of a holistic concept as described in [Penrose, 1995]. Figure 4.2 illustrates an impossible triangle. By virtue of what particular property is the triangle impossible ? The fact that it possesses this feature of impossibility is clear. It is supposed to be an image that conveys to the mind a particular

3-dimensional object, but that 3-dimensional object simply cannot exist in ordinary space. What is wrong with the picture ? Can we point to somewhere in the picture where the mistake was made ?

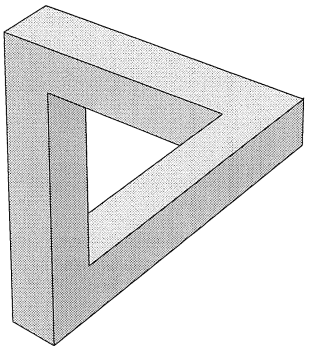


Figure 4.2: An impossible triangle.

The impossibility might be located in one specified corner of the triangle, so that if that corner is covered up the figure would make sense as the representation of a possible 3-dimensional object. It could be said then that the impossibility has disappeared. Apparently it did not reside in the rest of the picture. If any one corner is hidden, or any edge is removed, the structure suddenly becomes possible. When the attention is put on the complete picture, however, the impossibility re-emerges.

This impossibility is a property of the whole structure. It cannot be localised in some part : it is a feature of the complete object, so it is a *holistic* property. There are areas of mathematics dedicated to discussing this kind of property in a rigorous way.

4.3 Information Processing Abstractions

Perceptual systems should establish a set of information processing abstractions to allow the execution of perceptual steps hierarchically. The abstractions will characterise the representations and processes needed to solve the problem.

4.3.1 Matching Process

One example of processing abstraction in perception can be taken from the pattern recognition problem. The most immediate method of pattern recognition is to compare a pattern with stored models of known objects and choose

the best match. This technique, known as *template matching*, is performed directly in the image space. Templates consist of images of known objects and no feature from the objects is extracted. The enthusiasm for the template matching approach stems from several reasons. First, the ability to recognise objects must depend on matching a description of the image with a representation in the memory. What form should this memory representation take ? A template approach is to store the collection of 2-dimensional views of known objects and attempts to find the best match with the current input image. This approach mostly fails because objects generally never present the same exact view twice [Hoffman, 1986]. For example in visual perception, changes in viewer perspective, distance, and so forth produce changes in an object's shape that doom the template approach to fail.

4.3.2 Feature Based Process

Almost all current theories make the following strong assumptions about the initial input module for machine perception. Complex patterns are initially analysed into components by a set of independent, feature-analysing systems. The perceptual process takes place after the features have been reassembled. According to [de Callataÿ, 1986], the power of this paradigm is strongly dependent on the availability of features that are invariant to the expected changes in the input pattern. The choice of features is problem dependent. However, the recognition or classification methods can be independent of the problem domain and need not be restricted to pictorial inputs. This theory has by now accumulated a remarkably large and varied set of supporting data. The main disagreement concerns the nature of the features extracted at the initial stages and how the later assembly into objects is achieved.

4.3.3 Holistic Process

The properties of parallelism and distribution have attracted adherents who feel that human memory has a holistic character – much like a hologram – and those adherents consequently react negatively to discrete symbol-processing theories because these compute the needed information from constituent parts and their relations. [Dreyfus, 1979] for example, argues that pattern recognition in humans does not proceed by combining evidence about constituent features of a pattern but uses a holistic process. Thus, Dreyfus looks to neural networks as vindication of the long-standing criticism of symbolic theories. Symbolicism performs recognition by sequentially computing intermediate representations, and neural networks are said to perform direct recognition, so their solutions are holistic.

The characteristics are especially attractive to those who believe that AI has

to be based more on brainlike architectures, even though a wide divergence is present about the degree to which directly modelling of the brain is considered appropriate [Chandrasekaran et al., 1988]. Although some of the theories explicitly attempt to produce neural-level computation structures, other propose an intermediate subsymbolic level between the symbolic and neural levels [Smolensky, 1988]. The essential idea of these theories is that the neural connections define the information content rather than the representation of the information as a symbol structure.

4.4 Experimental Insights into Holistic Perception

The pattern recognition problem could be a good example to illustrate the implementation of holistic processes. Perceptual images, such as visual images, in part can represent the system interaction with its environment. The analysis of the physics of this interaction can be done by developing the understanding of these images. In most cases, the processing algorithm should recover the information about the 3-dimensional world from images.

4.4.1 Touch Perception as a Holistic Process

The sense of touch provides human beings with an important source of information about the surroundings. Because of its unique position at the interface between the human body and the outside world, touch sensing supplies sensory data which helps humans manipulate and recognise objects and warn for harmful situation [Russell, 1990]. Tactile sensing has the potential to fulfill a similar sensing role for robotic systems.

Tactile sensing produces images from contacts. As well as visual images, tactile images also contain 3-dimensional information represented by contact pressure distributions. Like in human systems, the descriptions of the tactile images may be developed through a holistic process, which directly maps the tactile pixels to a certain perceptual impression.

4.4.2 Experiment with a Robot Tactile Sensing System

The experiment is aimed at performing a classification process of tactile images, without the need to extract features from the images. A direct, featureless process to classify contact impressions of the gripped objects is investigated. This classification problem is considered to be a simple problem that allows the holistic process to be studied. Supervised and unsupervised learning neural network approaches will be evaluated. The implementation would rely on the generalisation abilities of the neural network parallel operation

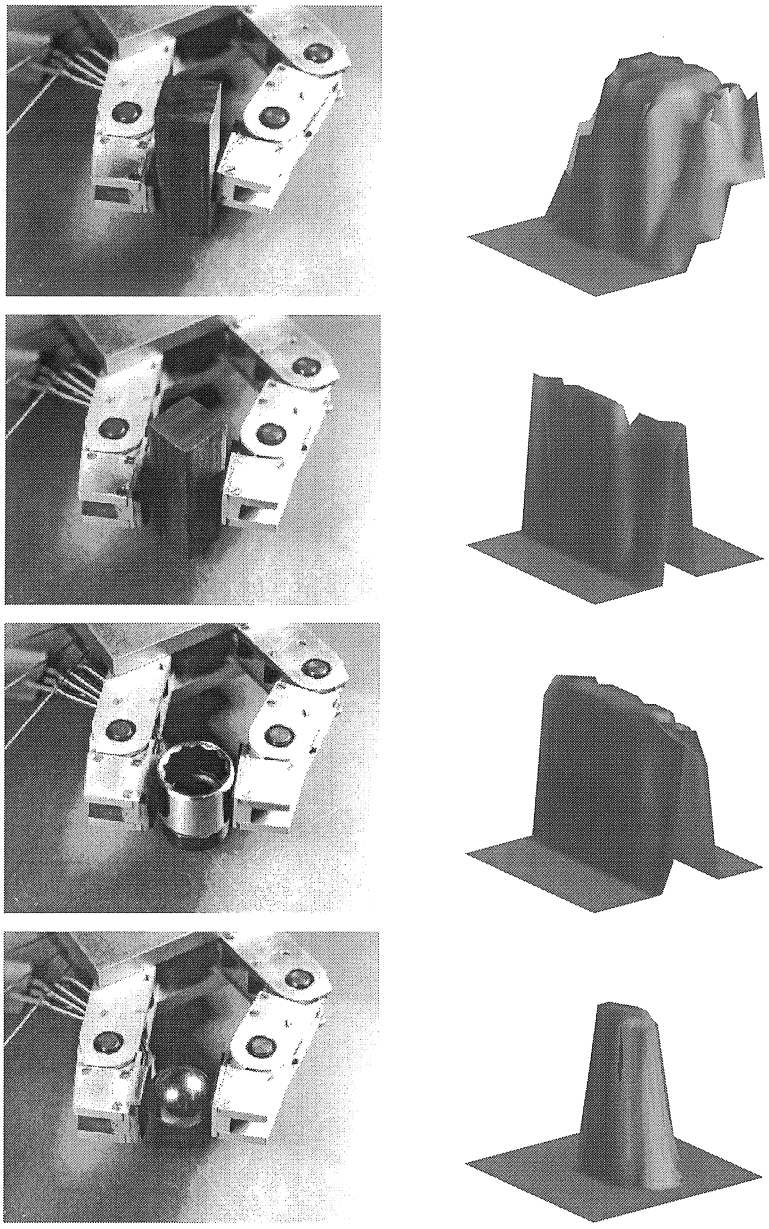


Figure 4.3: Different contacts of objects and the resulting tactile impressions represented by their contact pressure landscapes. Four different contacting shapes are shown from top to bottom : flat surface, edge, cylindrical surface and spherical surface.

and on its fault tolerant nature to achieve a holistic interpretation of the contact pressure landscape between the object and the robot finger.

4.5 Robotic Viewpoint of Tactile Sensing

Contact is one kind of interaction that an intelligent autonomous system can make during its presence in the world. The use of tactile sensors in a robot gripper provides the capability to monitor the contact with the gripped objects. Having information about the gripped object during the manipulation is considered to provide a potential enhancement to the robot manipulation behaviour.

4.5.1 Knowledge about the Manipulated Objects

In general, when manipulating an object, the configuration of a grasp is largely depending on the type of object to be manipulated [Reynaerts, 1995]. Also, the type of grasp can change considerably during manipulation. This creates the need for a manipulation planning system taking into account the above mentioned elements. This is a quite new element in manipulation science because until now, the grasp or at least the grasp type was supposed not to change during manipulation. Manipulation planning will start from a desired object movement to plan a sequence of manipulations. Although human manipulations are planned with an astonishing ease, manipulation planning is hardly mentioned in literature and clearly lies beyond the current frontiers of knowledge.

A basic parameter for performing manipulations is the local shape of the manipulated object. The local shape is the shape of the object surface which is in contact with the gripper. It can be identified from a single tactile impression obtained from the contacting object surface. In this experiment, four different classes of local shapes are considered. They are : a flat surface, an edge, a cylindrical surface, and a spherical surface, as shown in fig. 4.3.

The classical technique for object recognition is performed mostly by processing features which are extracted from the contact patterns between the object and the robot fingers. As a matter of fact, feature-based techniques have some intrinsic drawbacks. They mostly suffer from the amount of time it takes to complete the solution. Furthermore, the complexity contained in the patterns affects the representativeness of the features.

4.5.2 Tactile Sensing Characteristics

Some sensing characteristics must be considered when collecting data from tactile pixels (*"taxels"*). In general, the characteristics are mainly determined

by the type of the used compliant tactile transducer. The most important influencing feature is its non-linearity and the resulting hysteresis. Also, the occurrence of sensor noise brings a typical characteristic into the system. Any technique applied to interpret tactile signals ideally has to take the above characteristics into account.

In this experiment, a tactile sensor system based on a conductive rubber contact layer is used [Van Brussel and Belien, 1986]. The tactile surface consists of a 16-by-16 pixel matrix, and has a spatial resolution of 1 mm. The pressure resolution of each pixel is 64 levels ranging from 1 to 50 N/cm^2. The tactile sensor is incorporated in a two-fingered gripper system [Reynaerts, 1995] as shown in fig. 4.1. The tactile surface is mounted on the inner side of the distal phalanx of the gripper thumb.

4.6 Experimentation : Methods and Results

4.6.1 Objectives

Through the chosen classification task, some aspects of holistic nature will be investigated in the neural network classifiers. One important capability needed in performing a robust pattern recognition task is that the system has to be insensitive to the changes in the size, position and orientation inside the image to be recognised [Andrews, 1972]. Therefore, the investigation will be firstly focused on this aspect.

The second aspect to be investigated will be the featureless characteristic of the classification process. This investigation will be illustrated by a comparison between two classification approaches of the same set of images, one using the complete image frame as input and the other one using features extracted from the image as input. The final investigation will be on the performance comparison between the neural network classification techniques and the classical statistical pattern recognition techniques.

4.6.2 Direct Featureless Method

The idea of direct featureless classification is described as follows. Every classification process is performed based on a unique tactile image per contact. The conventional way to classify the object's local shape is mainly based on the features extracted from a corresponding tactile image [Reynaerts, 1993]. This method is depicted in fig. 4.4. As in the most of sensory systems, noise cancellation is usually firstly performed to suppress the non-zero initial values and electronic noise. Based on the resulting image, which is here called *fine image*, a feature extraction process is carried out. A classification method is then applied on the obtained features.

The proposed direct featureless classification is carried out by eliminating the feature extraction process. In this way, the classification is performed directly from the fine images. A neural network is applied as the classifier. By using this method, the processing flow layout inside the dashed-rectangle shown in fig. 4.4 is replaced by the layout shown in fig. 4.5.

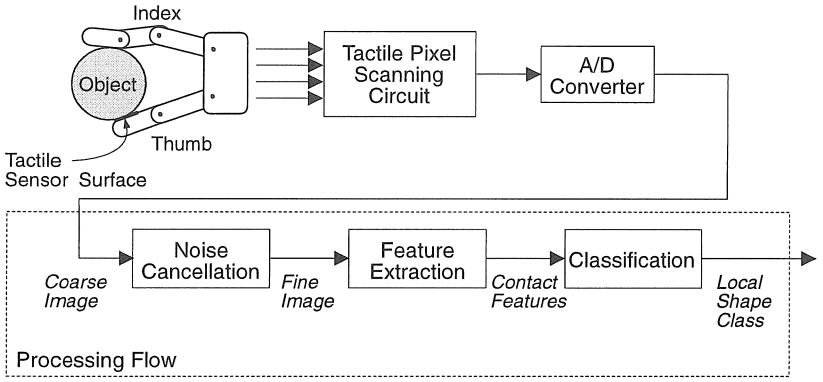


Figure 4.4: The scheme of tactile sensing image acquisition, followed by a conventional classification method of local shapes based on contact features.

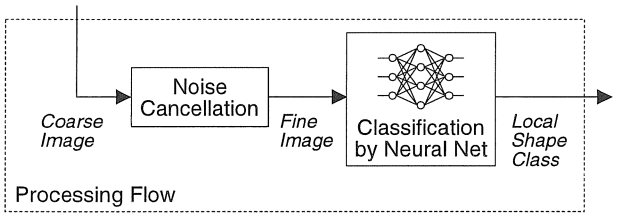


Figure 4.5: The classification method using neural network based on the obtained fine image.

4.6.3 Neural Network Paradigms for Pattern Classification

The taxonomy of neural networks that can be used for classification of static patterns is firstly divided between networks with binary and continuous-valued inputs. Below this division, networks are divided again between those trained with and without supervision. Networks trained with supervision which are used as classifiers, are provided with side information or labels that

specify the correct class for new input patterns during training. Networks trained without supervision, such as the competitive learning type networks, are used as vector quantisers or to form clusters. No information concerning the correct class is provided to these networks during training.

The above two learning methods will be investigated in this experiment. A supervised learning method implemented on a back-propagation network and an unsupervised learning method implemented on a learning vector quantisation (LVQ) network are applied to the specified classification problem[1]. The wider descriptions of these neural network paradigms can be found in appendix A.

Back-Propagation Networks for Pattern Classification

Back-propagation networks have been successfully used for classification tasks with performance similar to that of conventional non-parametric pattern classifiers such as k-nearest neighbour classifier with the benefit of inherent parallelism of the neural network structure [Sethi, 1991]. This type of networks have a capability of performing the classification of linearly non-separable patterns [Vandewalle, 1993a, Sethi, 1991], by which these networks are finding increasing use as non-parametric pattern classifiers. One difficulty often encountered in the successful application of back-propagation is the necessity of matching the network topology, i.e. the number of layers as well as the number of neurons in each layer, to a specified problem. The network topology is an important factor that can significantly affect the learning time as well as the overall input-output mapping performance as indicated by many experimental studies [Huang and Lippmann, 1987, Gorman and Sejnowski, 1988].

LVQ Networks for Pattern Classification

Unlike generally in competitive learning type network design which has no mechanism to dictate whether two arbitrary input vectors are in the same class or in different classes [Simpson, 1990], LVQ networks, on the other hand, have a capability to allocate reference vectors to the centroids of the decision regions without any *a priori* information concerning the distribution of data. In this way, a data vector is practically replaced by the index of the output neuron that it fires. It makes LVQ networks able to learn to classify input vectors into target classes chosen by the user. The target classes are introduced together with the corresponding input patterns during a training process. The LVQ strengths include its ability to perform non-parametric pattern classification and provide real-time nearest-neighbour response [Simpson, 1990].

[1] A software package - MATLAB - with the neural network toolbox, produced by The MathWorks, Inc. is used.

Unlike the feedforward multilayer paradigm, LVQ networks can classify any set of input vectors and not only linearly non-separable sets of input vectors [Simpson, 1990]. The only requirement is that the competitive layer must have enough neurons, and each class must be assigned enough competitive neurons.

Neural Network Input-Output

Both applied neural networks are introduced with the same size of input patterns. For the direct featureless method, the input vectors must consist of components as many as the number of tactile sensor pixels. For the used tactile sensor system, the input vectors is 16 $\times$ 16, or 256. Thus, the input layer in each network must have 256 neurons. In the case of feature-based classification, the network input is the applied feature vector. Thus, the number of input neurons equals the number of the applied features.

The number of neurons in the output layer of both networks is determined by the desired number of classes. Consequently in this classification problem, 4 neurons are required in the output layer of both networks. Each active neuron — a neuron outputting 1 — indicates the presence of an existing class. For the four local shape classes, the following output vectors are used : [1 0 0 0] to represent flat surface, [0 1 0 0] to represent edge, [0 0 1 0] to represent cylindrical surface and [0 0 0 1] to represent the spherical surface classes.

4.6.4 Noise Cancellation Technique

The first common step in the sensory signal processing is usually dealing with the treatment of undesired signal values coming from the sensory transducing process. In that concern, here, a set of image processing procedures is carried out first to the *coarse image* : the image obtained directly from the sensor reading hardware, in order to filter out the undesired non-zero initial values and electronic noise. The procedures and their sequence are described in the following steps.

1. A soft low-pass filter is applied to the coarse image. A 3x3 convolution filter **F** is used to eliminate noise. This matrix is defined in equation 4.1 :

$$\mathbf{F} = \frac{1}{18}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 10 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.1}$$

   The matrix component values are optimised experimentally.

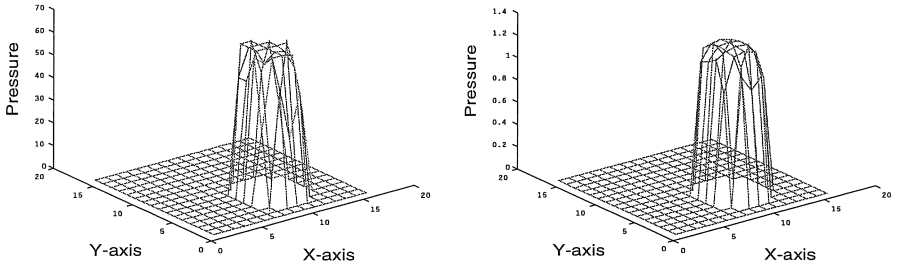
2. Next, a thresholding procedure limits the increasing pixel average values introduced by the low-pass filter. The threshold value was selected from experiments and 20% of the maximum pixel values is found to be the optimum value.

3. Finally, an arctangent function is applied to suppress the image irregularities. The function is expressed in equation 4.2 :

$$fine(x,y) = tan^{-1}\left(\frac{input(x,y)}{k}\right) \tag{4.2}$$

where $fine(x,y)$ and $input(x,y)$ are consecutively referred to as the fine image which is the resulting image of this step and the input image which is the image introduced to this step. k is a constant where its value is obtained from experiments. This value determines the degree of the irregularity suppression.

The suppression of image irregularities does not reduce the information contained by the image since the irregularities are more due to the sensor transducing and interfacing effects than to the occurring contact stimuli.



(a) coarse image (b) fine image

Figure 4.6: An example contact image of a spherical object showing the effectiveness of the noise cancellation technique.

The result of the whole noise cancellation technique is illustrated in fig. 4.6 for a contact of a spherical object. The next steps will use only the resulting fine images.

4.6.5 Featureless Neural Classifications with Image Uncertainty

Besides influences caused by the tactile sensing characteristics discussed earlier, the contact images also contain some uncertainties due to the non-uniform contact position and orientation measured at different gripping situations. The first experiment will be devoted to evaluating whether the neural network classifier can cope with this image uncertainty. The evaluation is done based on a test set consisting of 60 contact images. The four local shapes selected earlier, are represented in the test set. The images are obtained by gripping the sample objects under a certain average gripping force and variations in contact position and orientation are made.

The back-propagation network is trained by using a training set consisting of another 60 contact images which are paired with their corresponding class labels. The training set images are collected under a certain average gripping force and contact variations as well.

The LVQ network is firstly introduced to a number of target images representing the "ideal" contact position and orientation of each local shape. The ideal contact image refers to the desired proper position and orientation that occurs while gripping an object. The number of ideal images is ideally 1 for every class. Thus in this classification case, the LVQ network is ideally to be introduced to 4 target images.

Test Set Class	Shape Classified				Success Rate
	flat	edge	cylindrical	spherical	
Back-propagation network					
flat	14	1			80%
edge		10	3	2	
cylindrical	2	3	10		
spherical		1		14	
LVQ network					
flat	14	1			78.3%
edge		9	4	2	
cylindrical		5	10		
spherical		1		14	

Table 4.1: Network classification performances on images with uncertainty.

The classification results on the test set are presented in table 4.1. Both networks achieve a 100% success rate on the training set. For the back-propagation network, the optimal network topology is found experimentally as a network with 1 hidden layer consisting of 16 neurons.

For the LVQ network, the number of ideal images needed to handle the

classification is found from the experiments as of 60 images (in fact all images in the training set). The optimal LVQ network that achieved the above result also needs 60 neurons in the first layer. From those facts, it can be assumed that the LVQ network treats each image as belonging to a different natural class. This phenomenon demonstrates that the LVQ network is very sensitive to the variations in orientation and position. By comparing the success rates of both networks, even the back-propagation network seems more generalised in handling the images with uncertainty.

4.6.6 Featureless Neural Classifications with Image Certainty

It is already discovered by many researchers [Kuncheva, 1988, Davis, 1991] as well that neural networks are very sensitive to the changes in the size, position and orientation inside the observed image. Neural networks work well if there is not too much information about position and orientation contained in their input images. Until a certain limit, neural networks can generalise the inputs with some translational and rotational variations. If the variations are too large, they fail. For this reason, most of them either work with a set of constraining assumptions about the treated images or use some kind of preprocessing to remove the undesired information that complicates the classification task of the neural networks.

The experiment is now focused on classifying images which are free from the information about position and orientation of the contact pattern with respect to the tactile sensor surface. This is achieved by performing an input preprocessing technique before the neural network processing.

Neural Network Input Preprocessing

The technique basically computes the position and orientation of the object and then performs a translational transformation to place the object in the middle of the image frame and a rotational displacement so that all images have a certain, same orientation. In this way, the images do not lose their discriminatory clues regarding the shapes that are present before the transformations take place, such as pressure distribution.

The image transformations are basically performed by transforming the indexes of the image pixels. The image translational transformation is expressed by the following equation :

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i^* + \Delta y \\ j^* + \Delta x \end{bmatrix} \tag{4.3}$$

where i and j refer to the pixel row and column indexes in the new image respectively and i^* and j^* to the pixel row and column indexes in the trans-

formed image. Δy and Δx represent the linear displacement values, obtained by calculating the difference between the current image centroid and the centre of the image frame. The image centroid is computed by using the following equations :

$$j_c = \frac{\Sigma j_n}{A} \quad \text{and} \quad i_c = \frac{\Sigma i_n}{A} \tag{4.4}$$

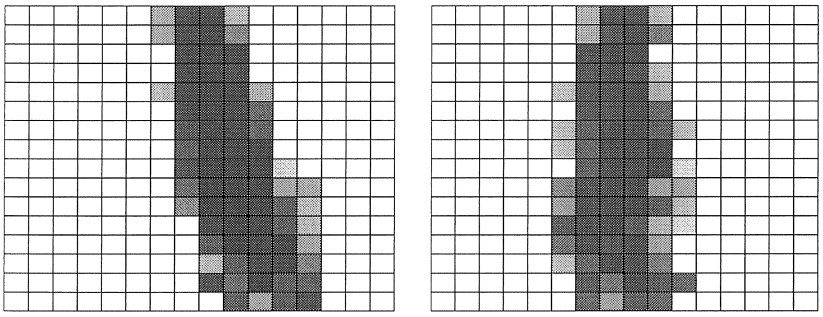
j_c, i_c are the column and row indexes of the centroid, x_n and y_n are the column and row indexes of the n^{th} active pixel, and A is the number of active pixels. The active pixels are the impressed pixels or pixels whose values are non-zero.

The image rotational transformation is expressed by the following equation :

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} cos\gamma & sin\gamma \\ -sin\gamma & cos\gamma \end{bmatrix} \begin{bmatrix} i^* \\ j^* \end{bmatrix} \tag{4.5}$$

γ represents the rotational displacement value, obtained by calculating the difference between the current orientation of the contact pattern and the desired orientation. The contact pattern orientation is defined as the direction of the principal axis of the contact area, and calculated by using the following equation :

$$\theta = 0.5 \cdot tan^{-1} \frac{-2\Sigma(j_n - j_c)(i_n - i_c)}{\Sigma(i_n - i_c)^2 - \Sigma(j_n - j_c)^2} \tag{4.6}$$



(a) before preprocessing (b) after preprocessing

Figure 4.7: Two-dimensional images from a contact of a cylindrical object, before and after preprocessing. The darkest colours represent the highest pressure levels applied to the pixels.

θ is the angle between the minor principal axis of contact area and X-axis of the image (parallel to the tactile matrix row), j_c and i_c are the location of the centroid, j_n and i_n are the column and row coordinate of the n^{th} active cell.

The transformations applied to an image will result in a number of pixels having no value. To those pixels, a zero value is assigned. The effect of the input image preprocessing is illustrated in fig. 4.7.

Classification Performances

The classification results on the test set of preprocessed images are presented in table 4.2.

Test Set Class	Shape Classified				Success Rate
	flat	edge	cylindrical	spherical	
Back-propagation network					
flat	14	1			91.7%
edge		14	1		
cylindrical		3	12		
spherical				15	
LVQ network					
flat	14		1		93.3%
edge		12	3		
cylindrical			15		
spherical				15	

Table 4.2: Network classification performances on preprocessed images.

Both neural networks achieve a 100% success rate on the training set. For the LVQ network, classification on the test set is achieved by using 1 ideal image for each class (so that 4 ideal images in total) and 4 is found experimentally to be the optimum neuron number in layer 1. It means that any input image on the test set can be clustered into one of the classes represented by the ideal images. This shows more clearly the clustering characteristic of a competitive layer.

The classification performance of the back-propagation network is also improved compared with its performance on images with uncertainty. But here, compared with the LVQ network, the back-propagation network does not perform better. This comparison demonstrates that a classification process merely relying on the generalisation ability (on multilayer feedforward type networks) does not perform better than a classification relying on the clustering ability (on competitive learning type networks). In fact, the nature of multilayer feedforward network employed by the back-propagation algorithm is

dedicated to learn the mapping of input-output relationships, without knowing the correlations among inputs. Multilayer feedforward networks have no capability to capture the natural classes among the similar input vectors. So, the classification done by the back-propagation network is more to accomplish a generalised image mapping to the image's class labels.

In general, although the result presented in table 4.2 can be considered as good, it is still far from 100%. One reason is that the digital images used are actually digitalised spatially and are not represented as continuous functions that can be rotated and translated precisely. In that case, there exists a degree of uncertainty when translating and rotating a digitalised image. By these translational and rotational transformations, pixels of a transformed image cannot be precisely laid on the tactile pixel grid. Some numerical truncation to the pixel index numbers would have to be done. So, a complete transformation of the information inside the image related to its position and orientation is not obtained. This phenomenon is particularly due to the low sensor resolution.

4.6.7 Feature-Based Neural Classifications

In order to investigate how a contact image can be represented by features and how neural network classification processes can be carried out based on these features, the following experiment is done. The classifications are based on two features, which are selected from experiments to optimally represent the tactile contact situations.

Feature Extraction

The most important criterion for the feature selection is that the feature must retain much of the useful discriminatory information present in the original data. In this experiment, two features are used. The first feature is based on a histogram representation and the second feature represents the total pressure values applied to all pixels.

To extract the first feature, two histogram bars are used to represent the contact pressure distribution information contained in the tactile images. It is assumed that different types of contacts would have different distributions and absolute values. A threshold value is used to group the image pixels into two histogram bars, called $High\_hist$ and Low_hist, according to their pressure levels. The selection of the threshold value is very critical since steep slopes are present along the edges of the images. It is found by experiments that 99% of the maximum pressure level is a good criterion to distinguish the images. The thresholding determining which histogram an image pixel $img(i, j)$ belongs to is expressed by the following equations :

$$h(i,j) = \begin{cases} 1 & \forall i,j \mid img(i,j) > 0.99 \cdot \max(img(k,l)) \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$l(i,j) = \begin{cases} 1 & \forall i,j \mid img(i,j) \leq 0.99 \cdot \max(img(k,l)) \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

$h(i,j)$ and $l(i,j)$ in fact label all pixels with their corresponding histograms. The values of the histograms are then calculated by the following equations :

$$High\_hist = \sum_i \sum_j h(i,j) \tag{4.9}$$

$$Low\_hist = \sum_i \sum_j l(i,j) \tag{4.10}$$

$High\_hist$ is the one that is used as the feature in the classification. The other feature used is the total of all pixel values, which is called here $Volume$ and calculated as follows :

$$Volume = \sum_i \sum_j img(i,j) \tag{4.11}$$

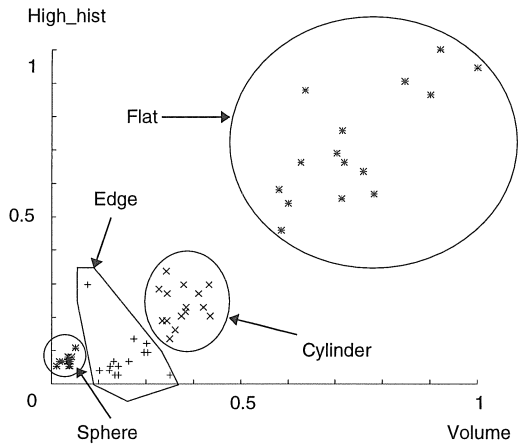


Figure 4.8: The dimensional feature space which used $Volume$ and $High\_hist$ as the features. The distribution of the training set, which is used as the sample data, is shown clustered into four existing classes.

By extracting these two features, the contact image space is transformed to a 2-dimensional feature space, on which the feature-based classification methods are based. The previous training set data is used as the sample data, and the feature space of this sample data is shown in fig. 4.8.

Classification Performances

The classification performances are presented in table 4.3. From the experiment, the optimal back-propagation network topology is found as the one with 1 hidden layer consisting of 16 neurons, and the optimal LVQ network topology is found consisting of 4 neurons in the first layer. Both networks achieve 98.3% success rates on the training set.

Test Set Class	Shape Classified				Success Rate
	flat	edge	cylindrical	spherical	
Back-propagation network					
flat	15				98.3%
edge		14	1		
cylindrical			15		
spherical				15	
LVQ network					
flat	14		1		96.6%
edge		14	1		
cylindrical			15		
spherical				15	

Table 4.3: Feature-based neural network classification performances on images with uncertainty.

4.6.8 Statistical Pattern Classifications

In order to have a better evaluation, the same test set images are introduced to feature-based classification processes performed by two statistical pattern classification techniques : nearest-neighbour rule and parametric Bayes rule (see appendix B for the descriptions of the rules). The classification processes are based on the two features which were used in the previous neural network classification. The classifications performed by the two methods yield the results presented in table 4.4. It is shown that even with feature-based statistical pattern classification techniques, a 100% success rate is difficult to achieve.

Test Set Class	Shape Classified				Success Rate
	flat	edge	cylindrical	spherical	
nearest-neighbour rule					
flat	14		1		96.6%
edge		14	1		
cylindrical			15		
spherical				15	
Bayes rule					
flat	15				96.6%
edge		14	1		
cylindrical			15		
spherical			1	14	

Table 4.4: Statistical pattern classification performances on images with uncertainty.

4.7 Interpretation of Experimental Results

In general, the success rates of the feature-based classification is better than the success rates of the featureless ones. Simply, feature-based neural classification has a better performance. Viewing it more carefully, these performances are not alone thanks to the non-image input, but also determined by the kind of features to be used. Compared with the feature-based statistical classification, the main advantage of performing the non-feature-based classification is that the crucial step to select and extract the contact features can be omitted.

METHODS	Execution Time (*msec*)
Direct Featureless (*using preprocessed input*)	
Back-Propagation network	8
LVQ network	10
Feature Based	
Back-Propagation network	70
LVQ network	80
Nearest-neighbour rule	80
Bayes rule	83

Table 4.5: Comparison of the required classification time for 1 image, based on executions performed on a HP 712/80 workstation.

Another advantage can be detected in terms of classification times. A rough comparison of computing times consumed by all evaluated methods

is presented in table 4.5. It is assumed that by looking at the needed execution time (outside the neural network training time, the time for feature sampling in the statistical classification methods, and the input preprocessing time) the direct featureless neural network method is more straightforward in performing the interpretation of the contact situations.

The effectiveness of the input preprocessing in the neural network featureless classifications shows that the neural networks do not treat the input as merely a set of statistical data, but more as a set of images. One disadvantage demonstrated by the need of the input preprocessing is that the neural network classifiers still need a fix viewpoint to observe the discriminatory clues regarding the shapes that present in the coarse images.

4.8 Conclusions

A direct featureless neural network classification method applied to tactile contact impressions is presented. Compared with the more classical feature based methods, the proposed method is more direct and straightforward, which is in fact reflecting the holistic nature of neural network processing. On the other hand, the requirement of network input preprocessing shows that the evaluated neural network classifiers are not insensitive to pattern variations in position and orientation. The latter remains a problem for creating a good model for artificial neural network.

The result on the experiment using image with certainty in position and orientation demonstrates that neural network approaches interpret the input as a complete image, and not as a set of statistical data of pixel values. This evaluation can lead to a conclusion that by design, the neural network model already contains the nature of parallel operation, and that can be a good platform to implement the holistic perception.

The experimental case study demonstrated here deals with a single sensory information source with data in the form of images. A more interesting holistic phenomenon can apparently be found in applications where more than one sensory information source are available.

Chapter 5

Learning Reflexive Action : Connecting Vision to Motion

Don't ever think about it.
Just do it !

NIKE AD.

5.1 Introduction

One important sensory-motor activity in living biological systems is reflex movement. Reflex movement, as distinguished from voluntary movement, is a kind of action that a human or an animal does not intentionally plan to do when encountering a potential painful or harmful experience.

From the physiologic viewpoint, the reflex is a unit reaction in the central nervous system which has a stereotypical and repeatable input-output relationship [Ito, 1984]. The quick reaction to the stimulus is obtained through a crossed connection within a given segment of the spinal cord that allows the receptors in the periphery to produce coordinated muscle contractions with no added input from higher motor levels.

In practice, many of the desired robot reactive capabilities should possess the characteristic of this kind of natural control scenario, in which the perception-to-action connection is set very closely. The term *reflexive behaviour* must be distinguished from the term of *reactive behaviour*. Reactive behaviour is

prevalent in the planning domain while reflexive behaviour is more used in the control domain, referring to the generation of direct responses. In the context of robotic control, reflexive actions can be described as the desired actions which are generated immediately from the input sensed data in a single inference step without considering explicitly the robot's goals, available actions, or their predicted consequences [Mitchell, 1990]. On the other hand, reactive actions include also the consideration of the robot's final goals, selection of possible actions, and the future consequences of current actions. The cockroach behaviour to flee in an appropriate direction from a source of wind (described in section 2.4.1) is one example of a reflexive behaviour. The directional responses of the cockroach's escape system are generated without any consideration of the future situations.

This chapter presents the development of a learning strategy to build a form of reflexive behaviour. Here, building the reflexive behaviour is referred to as the building of rules to infer the desired action immediately from the input sensed data in a single inference step. These rules are built through learning and implemented by a neural network controller.

5.2 Visual Tracking as a Reflexive Action

A visual tracking behaviour by a robot arm is chosen as a case study to illustrate reflexive behaviour. By definition, tracking implies a type of action that has to be executed in real time. Tracking demands more an instant reaction to the current situation than a longer planning process to satisfy a specified goal in the future. Some interesting aspects may also be studied from this case study. Besides investigating the eye-hand relationship model that can be built by the neural network through learning, the emphasis will also be put on the investigation of the gradual improvement of the behaviour, including the acquisition and the refinement of the skill.

A reinforcement learning approach is applied. This approach encourages the gradual improvement process of the behaviour without an explicit set of learning examples. The implementation of this particular reflexive behaviour relies on the reinforcement-comparison learning architecture (introduced in section 3.6.3). By using this learning algorithm, a behaviour to respond to a condition observed by the vision system, through some actions in the form of robot arm joint displacements is developed. The mapping between visual information and robot movements is a highly nonlinear relationship which depends on the camera-robot relationship and the robot kinematics. Neural networks may be used to learn the entire transformation implicitly, thus avoiding explicit computation of all intermediate transformations.

5.3 Overview of Human Visual Tracking Skill

As biological systems, human beings have a sophisticated skill to perform tracking. Humans approach the tracking task in a direct way. In reaching an object, a human adjusts the direction and posture of his/her arm without explicitly computing the eye-hand transformation. To estimate the required movement of every joint of the arms, humans mostly recall their past experiences from similar situations. The precise arm motion toward the object is then performed by watching the object all the way and making the necessary corrections to its path, and following and tracking the object if it is in motion, until he/she reaches it satisfactorily. This skill is refined gradually by repeated practice and by correcting deviations from desired behaviour. The more often and the more diverse situations have been encountered, the more brushed up skill would be performed.

Like most human sensory-motor coordination skills, acquiring concepts on how to perform these tracking tasks represents only the initial phase in developing the requisite skills [Hetherington and Parke, 1986]. Further on, the human tracking ability is developed through a skill refinement process. An illustration of this phenomenon is given by fig. 5.1

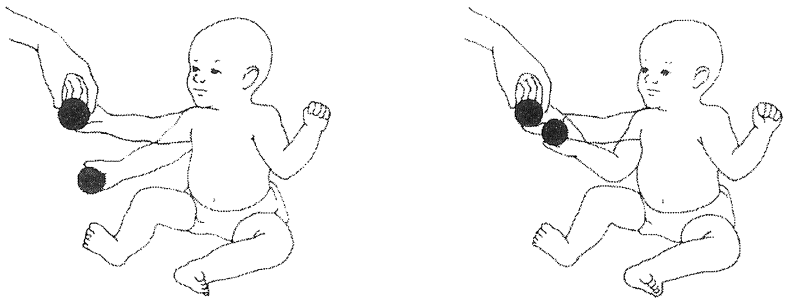


Figure 5.1: Evidence of the refinement of human visual tracking skills comes from a series of experimental visual-motor behaviour studies [Hetherington and Parke, 1986]. During the first half year of life, the hand and eye of normal infants in average environments are simply not well coordinated. At first, the infant may attend to an object placed in his/her visual field, but only in the second month can any swiping action be seen. As shown in the figure, in this phase the infant would misjudge the spatial distance of the object. Gradually the young infant learns to use his/her eye and hand together, so that he/she not only swings at the target but eventually can accurately and consistently contact the object with his/her hand.

In performing the refinement process, external cues are needed to guide the process towards the desired behaviour. In this sense, process reinforcements which are contingent on behaviour are required. From the psycholog-

ical viewpoint, reinforcement refers to a consequence that increases the likelihood of a behaviour occurring again. Although the term "reinforcement" suggests a positive consequence, it can be either positive or negative. A positive reinforcement is a "pleasant" stimulus that increases the likelihood of a response occurring again and a negative reinforcement is the removal of an "unpleasant" stimulus, thereby decreasing the likelihood of a response occurring again.

5.4 Overview of Robot Vision-Based Control

From a robotic point of view, visual tracking control has also become an attractive field. Much research on robot control with visual feedback has been carried out and many schemes have been proposed. Most of the approaches to robot control using computer vision require to convert the camera images of the scene to real world coordinates, which in turn must be converted to joint angles for the robot arm. Formally, this involves the solution of the inverse perspective projection and inverse kinematic equations of the system which have to be handled in real time. Another critical issue in performing a visual tracking task is that of the real time strategy to combine robot controllers and the vision systems. If vision is to be used as the controller's primary source of information about the location of the object relative to the robot end-effector, then a data rate appropriate to the dynamic performance of the arm is needed.

The task in robotic visual tracking control can be specified as :

> *Move the manipulator in such a way that the projection of a moving or static object is always at the desired location in the image.*

From that task specification, two distinct subtasks can be identified :

- Locating the object in the robot's workspace.
- Tracking the robot end-effector as it moves toward that object.

According to how the vision information is involved in the control loop, there have been two implementation approaches, as described hereafter.

5.4.1 Look-and-Move Control Approach

This approach [Weiss, 1984] comprises two phases as shown in figure 5.2. A "look" phase operates as an outer control loop using camera-derived information and applies the obtained information to an inner joint space "move" controller that relies on joint sensors for its feedback.

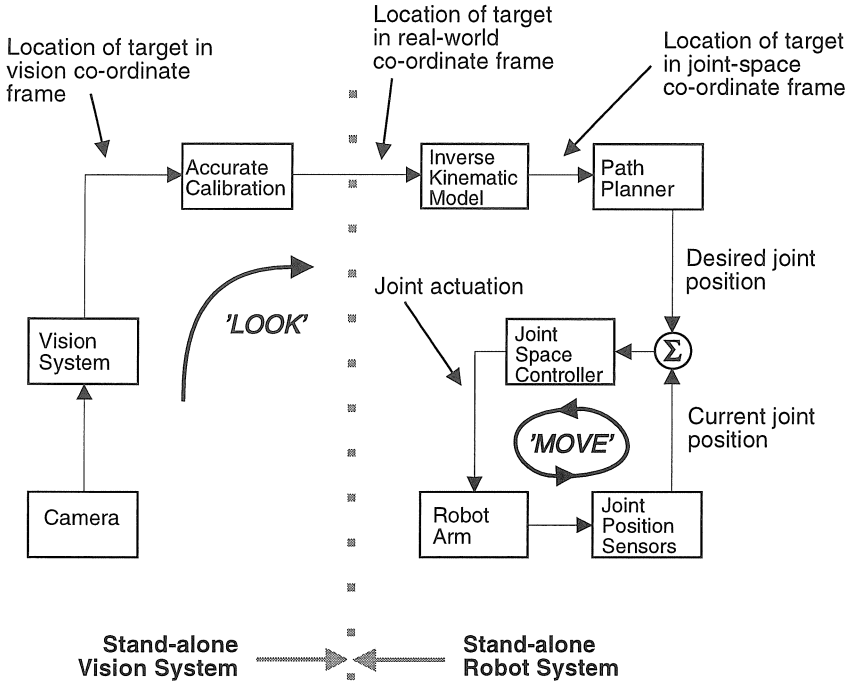


Figure 5.2: Phases in the look-and-move visual control approach.

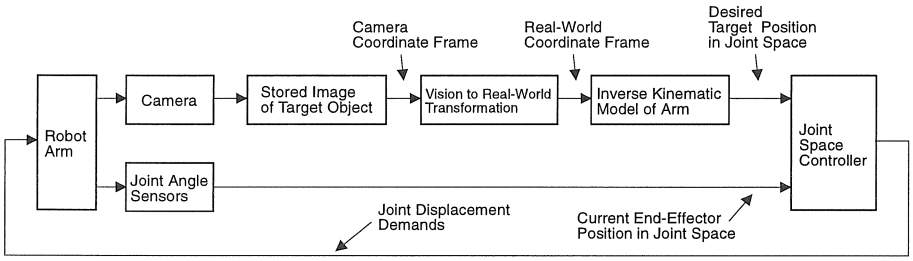


Figure 5.3: Look-and-move controller structure.

In the look-and-move approach, the vision system is used only to give the coordinates of the desired target position as illustrated by figure 5.3. Following calibration and conversion of the coordinate into real world units, this data is passed to the joint-based robot controller. The controller then uses joint sensors to evaluate its current position and, during the trajectory, compares measured and desired joint angles to compute the necessary joint torque demands.

The drawbacks of this approach is its demand of the accurate transformation and system calibration. An algorithm for accurately converting vision coordinate system into real-world coordinate system is required. A reliable algorithm for evaluating the set of joint positions and placing the end-effector at a specified point is needed as well as perfect joint position sensors.

5.4.2 Integrated Vision-Control Approach

Integrating a vision-based end-effector tracking scheme directly into the dynamic control feedback loop can potentially give greatly enhanced performance over that offered by a look-and-move approach. This approach gives significant advantages [Wijesoma, 1993] :

- Errors caused by vision system calibration are eliminated by using vision as the primary sensor system for locating both the target and the end-effector.
- Errors caused by poor estimates of the kinematic model parameters of the arm are reduced because the inverse model is used not only for evaluating the desired joint angles corresponding to the target position, but

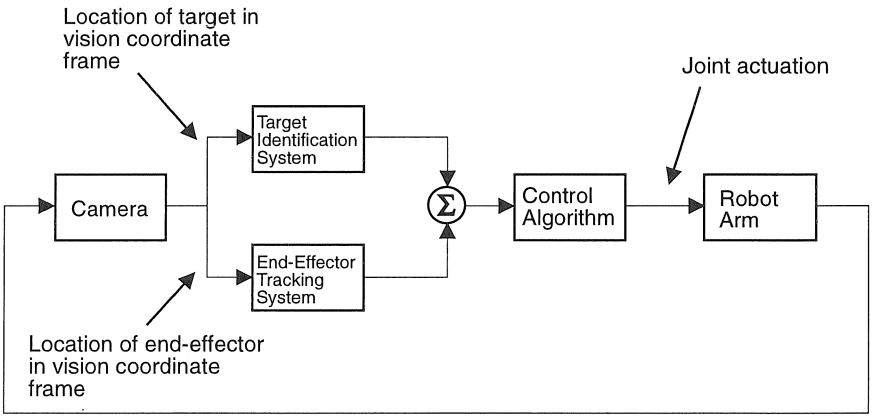


Figure 5.4: Integrated vision-control approach.

also to estimate the joint angles during motion that correspond to the end-effector position as measured by the camera.

The overall scheme, as shown in figure 5.4, can be compared with the look-and-move scheme shown in figure 5.2. Based on this scheme, two types of vision integration strategies into the robot control loop can be implemented [Wijesoma, 1993], as discussed hereafter.

5.4.3 Vision Integration into a Joint Space Controller

Here, as well as giving the desired target position, vision is used to track the end effector during the motion of the arm. This end-effector position is then used to derive an estimate of joint angles during motion by applying first the vision-real world transformation algorithm and then the inverse kinematic transformation. Control, as for the look-and-move case, is then achieved by comparing desired and estimated joint angles. Figure 5.5 illustrates the controller structure.

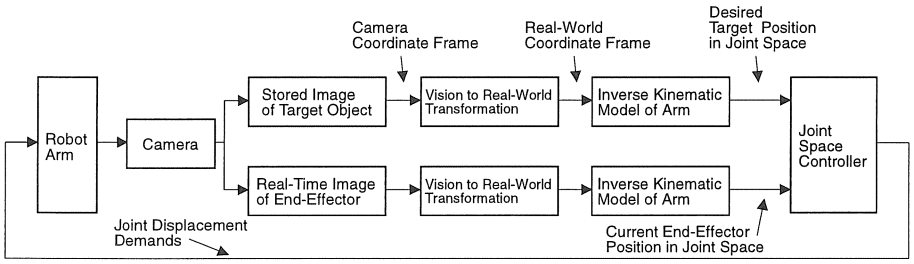


Figure 5.5: The structure of the vision integration with joint space controller.

5.4.4 Vision Integration into a Cartesian Space Controller

As in the integration into a joint space controller, here vision is used to derive the desired target position as well as the current end-effector position during motion. The cartesian space controller is used to evaluate the joint motion demand. Error between the current and desired position on the trajectory are evaluated as errors in cartesian space and not joint space.

The joint angle estimates derived via inverse kinematic transformation are only used by the controller to estimate the nominal dynamics of the arm at any time.

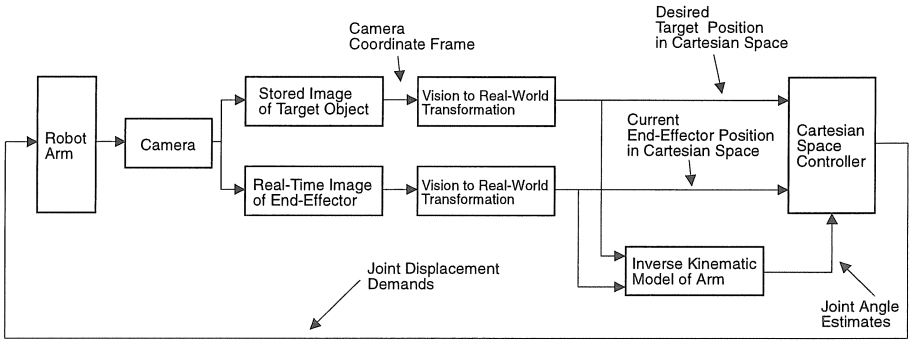


Figure 5.6: The structure of the vision integration with cartesian space controller.

5.4.5 Camera Integration Strategies

A significant difference in the applied control strategies can also be due to the placement of the camera. The camera may either be mounted on the end-effector of the manipulator, known as eye-in-hand configuration, or it may be statically located at a certain position relative to the robot workspace.

Eye-in-Hand Strategy

Here, the camera is mounted on the end of the robot arm and views the target directly. The task of moving to pick up an object then becomes one of moving the arm until the camera sees the desired object located under/in front of the robot end-effector. This type of camera integration strategy leads to what is called in control terminology : *visual servoing*. This approach gives an advantage of not being affected by the inaccuracy of robot arm kinematic model and robot joint sensors, but suffers from two principal disadvantages :

- The camera does not give an overview of the entire workspace and so cannot easily be used for the overall task of locating and identifying features within that workspace.
- In the case of dealing with static objects only : the image processing system must repeatedly relocalise the target object and must do so at as great rate as possible, which is computationally intensive, while in contrast the fixed eye strategy (described hereafter) only needs to localise the target position once. Though, in dealing with moving objects, both strategies need to repeatedly localise the target object.

Fixed Eye Strategy

This strategy applies a static camera base frame which allows to observe the entire robot workspace without any influence due to the robot motion. Here, the same camera view can be used both to gain an understanding of the robot workspace and as the primary source of information pertaining to the robot. The task of moving the end-effector to the object becomes one of moving the robot until the camera sees that the end-effector is at the related desired position. Applying this strategy to the look-and-move control approach, it introduces some major disadvantages as to the need for accurate robot arm kinematic models and robot joint sensors. On the other hand, applying this strategy to the integrated vision-control approach, it can therefore eliminate the need for accurate robot joint sensors, as the location of the end-effector is no longer determined by applying the forward kinematic transformation. This type of camera integration strategy leads to a domain called *hand-eye coordination*, which originally refers to the coordinated hand-eye activity in human beings. The eyes' positions allow humans to observe the entire arms' movements.

5.5 Previous Work on Robot Vision-Based Control by Neural Network

Significant work has been conducted in the domain of robot visual based control using neural networks. Most of it is dealing with mapping between object positions seen by the camera and the arm joint position configurations.

[Kröse et al., 1990] uses a back-propagation network to map the position of the target object in the camera field to the joint angle position displacements needed to make the end-effector reach the target object. The adaptation of the back-propagation network is done off-line using precollected training samples. A 6-DOF robot arm is used and the target object position is restricted on a 2-dimensional plane. In [Ritter et al., 1992], a 3-dimensional hand-eye coordination problem is presented, using visual information from a stereo camera system to control kinematically and dynamically a robot arm. A self-organising system is built by using a Kohonen network to map the 2-dimensional coordinates of the target object in the visual planes of the cameras 1 and 2 to the arm's joint position vector required to locate the end-effector. Here, a 3-axis articulated robot arm is used. [Kuperstein, 1989] has conducted similar work to realise the same type of hand-eye coordination with a back-propagation network. An automatic procedure is presented to generate randomly the required training examples.

Research has also been conducted in the field of visual servoing. A control scheme has been proposed by [Hashimoto et al., 1992b] to position and

orient the end-effector by using visual information from a single eye-in-hand camera system. The control system directly integrates visual data into the servoing process without decomposing the process into determination of the position and the orientation of the workpiece, and inverse kinematic calculation. This method is applied to a 6-DOF robot arm. Back-propagation networks are used for the determination of the change in the joint angles required in order to achieve the desired position and orientation. The visual information is introduced in the form of extracted image features. The same type of work is also done by [Torras et al., 1994]. The difference lays in the type of image features they used. Another important result on visual servoing is presented by [Pomerleau, 1989]. Here, a backpropagation network is used for the guidance of a mobile robot based on the visual information from a single camera. The neural network learns to map directly the camera images of the road ahead to the vehicle, to the steering directions of the wheels. A training process is performed by using a set of precollected training examples right before the real operation is done.

It is observed in this short overview that much of the related work is dealing with robot arm positioning problems. The commands to the robot are given in the terms of joint position values to achieve a certain static arm configuration. The information observed by the vision system is mainly representing a static position of objects in the robot workspace.

5.6 Reinforcement - Comparison Approach to Visual Tracking Control

The strong requirement of a fast response in the tracking control problem can be approached by performing a straightforward mapping between the current situation and the appropriate action. Since the criterion for the tracking task is to move the robot hand towards and to reach the object, every single robot action that fulfills that criterion will be highly rewarded. The mapping from the current sensory inputs to the robot movements to perform the tracking tasks can be identified as a problem to obtain a high immediate reward, without really needing to optimise the long-term future rewards. Tracking behaviour should be built without any expectation of deterministic object movements. It makes the learning system not having any future state to be optimised, instead of the immediate reward specified by the criterion itself. In this case, there is no need to optimise the total reward in the long run. The reinforcement-comparison learning architecture is suited for this kind of problem (as described in section 3.6.3).

5.6.1 Tracking Task Description : The Case Study

As the case study for the learning visual tracking problem, a 2-dimensional tracking task carried out by an articulated 6-DOF robot arm is chosen. This case study is limited to the tracking control problem of target object motions with a constant velocity. The target object moves in a horizontal plane within the robot workspace. A camera is mounted in the robot gripper and set to face downward to the object, as shown in fig. 5.7.

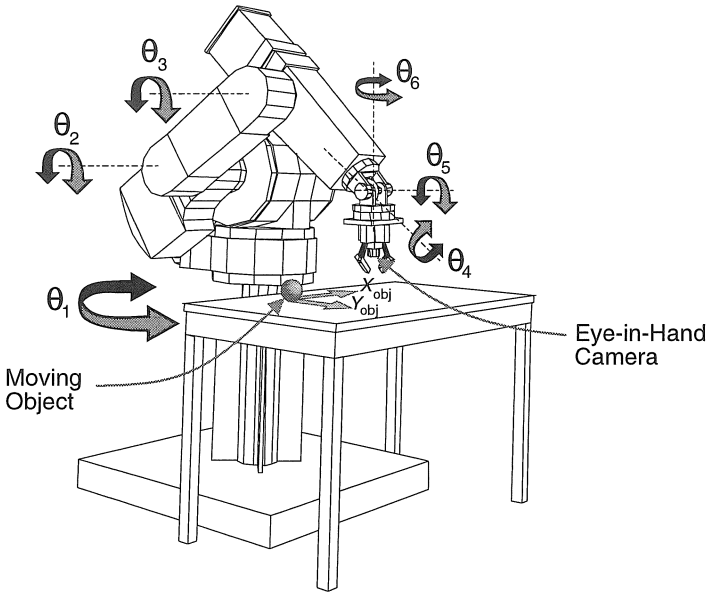


Figure 5.7: The 6-DOF robot arm set-up to perform the tracking task. A camera mounted in the gripper is facing downward at a constant height. The object moves in a horizontal plane.

The robot tracks the moving object at a constant height. The vision system localises the object during its motion. From this visual localisation, the relative state between the robot end-effector and the target object can be identified, and fed to the controller. The robot control command is given at the joint level. The consideration to send the action command at this level is to mimic the way of biological motoric systems, in which the kinematic and dynamic models are not explicitly included.

According to the more classical visual control approaches described previously, the tracking controller applied in this case study can be categorised

as the vision integration into a joint space controller, in which an eye-in-hand camera is used.

5.6.2 Specifying System's State and Action

The relative state between the robot end-effector and the target object obtained from the vision system is used as inputs for the controller. Basing the control merely on the object positions is not sufficient to perform a tracking task. The knowledge about the object velocities is normally used to perform a good tracking performance [Katupitiya, 1985, De Schutter, 1986].

From each situation resulting from a single action execution, the vision system observes the following world state $\vec{X}(t)$:

$$\vec{X}(t) = (\Delta x, \Delta y, \dot{\Delta x}, \dot{\Delta y}) \tag{5.1}$$

where $(\Delta x, \Delta y)$ is the object's relative position with respect to the robot end-effector and $(\dot{\Delta x}), \dot{\Delta y})$ is the object's relative velocity with respect to the robot end-effector, obtained by differentiating the object's relative position.

The action generated by the controller is specified in the terms of robot joint states. Since the robot tracks the moving object at a constant height, only two robot joints : joints 1 and 2 are significant to be controlled. The movement of other robot joints are determined as a function of the movement of joints 1 and 2. The complete robot movement is then expressed in the following way :

$$\vec{ee}_{vel} = f(\dot{\theta_1}, \dot{\theta_2}) \tag{5.2}$$

where $\vec{ee}_{vel}$ is the resulting end-effector linear velocity in the horizontal plane; $\dot{\theta_1}$ and $\dot{\theta_2}$ are the velocities of the significantly controlled joints 1 and 2.

5.6.3 Reward Predictor with Dynamic Model of the Environment

As shown in fig. 3.11, the two neural networks constructing the reinforcement-comparison learning architecture : policy and reward predictor networks are adapted over time. The policy network outputs the actual control signals while the reward predictor network guides how the action network is adapted.

Based on the above basic reinforcement-comparison learning architecture, an adapted architecture for the visual tracking control is implemented, as depicted in fig. 5.8. The reward predictor network inputs a description of the current world state $\vec{X}(t)$ and outputs a single number $R^*(t+1)$ (refers to predicted *reward*), which is a prediction of how well the action network is

doing in creating a "good" next situation based on $\vec{X}(t)$. To make this system work, learning rules are needed for both action and reward predictor networks. Those networks are trained separately.

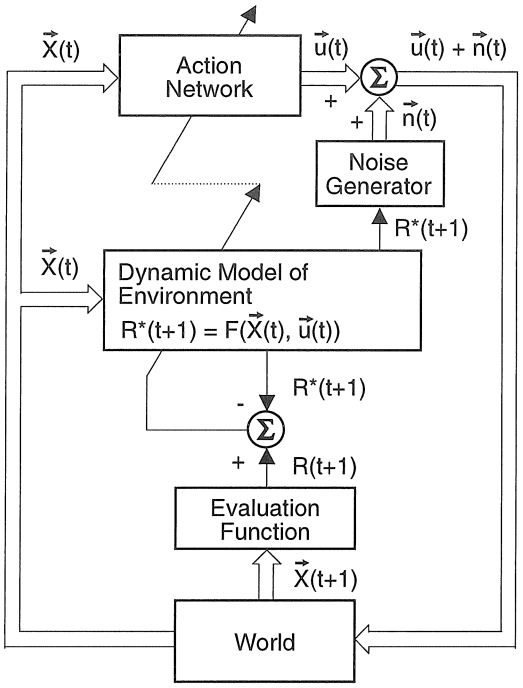


Figure 5.8: The adapted reinforcement-comparison scheme for the tracking task.

To accomplish the reward prediction process, reward values are specified to represent the possible resulting situation qualities. These reward values should be spread along the range of the possible qualities. A problem often occurs if rewards are not distributed around a baseline of zero, but around some other, unknown value. Worse yet, the baseline may change from state to state. A low reward value in one state may be the highest attainable in other [Sutton, 1991]. To handle such variations, the baseline value must be dynamic.

[Werbos, 1990] proposed an adaptive reward system that uses a neural network as a model of the dynamics of the environment, applied in a system called back-propagated adaptive critic (BAC). In BAC system, a dynamic prediction about the effect of the current control signal is made by using the dynamic model of the environment.

In this implementation, the use of a model of the dynamics of the environ-

ment as an adaptive reward predictor system is adopted. The model of the environment is built in the reward predictor network through learning (for short, further on, the reward predictor network is called prediction network). The model of the environment outputs a prediction of the new world state, as a consequence of the control signal $\vec{u}(t)$ produced by the action network based on the input $\vec{X}(t)$. The predicted state is determined by the dynamic model of the environment F represented by :

$$R^*(t+1) = F(\vec{X}(t), \vec{u}(t)) \tag{5.3}$$

where $\vec{u}(t)$ refers to the vector of action at time t and $\vec{X}(t)$ is a vector describing the current state of the world.

The prediction network is trained to model the relationship between the total control signal that affects the new situation (noise-added control signal) $\vec{u}(t) + \vec{n}(t)$ produced at $\vec{X}(t)$, and the quality of the resulting world state $R(t+1)$ judged by the evaluation function. This relationship in fact models the environment F and is to be expressed as follows :

$$R(t+1) = F(\vec{X}(t), \vec{u}(t), \vec{n}(t)) \tag{5.4}$$

where $\vec{n}(t)$ refers to a vector of random noise values.

5.6.4 Action Exploration Policy

The role of the policy network is performed by a neural network, the so-called action network, together with a noise generator. Noise is added to the output of the action network in order to make the system generate stochastic actions, to explore the possible action space. Noise makes the system respond differently to the same input conditions. Yet, once the system learns a suitable action for an input condition, the system is expected to perform the same action every time this input condition occurs. The system cannot perform the same action if noise is always present. As long as noise is present, the system's actions are not consistent.

To enable the system to perform consistent actions for the same input conditions, the noise has to be gradually reduced as learning proceeds. On the other hand, if the noise is reduced uniformly for all input conditions, the system's adaptability is logically becoming low for conditions which are not yet well learned (or which have never been encountered before). This is because the noise reduction reduces the exploration space of the suitable actions. It is thus necessary to vary control of the noise strength for each individual input condition.

To control the noise strength for each input condition, the system has to evaluate whether the action network output can respond with a suitable action for the current input condition before noise is added to the network output.

This implies that the system has to predict the evaluation result that will be obtained if the system acts without noise. The system has to control the noise strength according to this prediction. When the evaluation of the action without noise is predicted as suitable, the noise should be reduced.

The noise control process is established by means of the reward prediction mechanism. The amount of noise is controlled based on the prediction of the next world state, performed by the dynamic model of the environment. The total output then affects the next world state $\vec{X}(t+1)$.

The next world state $\vec{X}(t+1)$ is then evaluated by an evaluation function, that produces the $R(t+1)$ signal, a judgement of the action quality. The goal of this control scheme is to find a vector $\vec{u}(t)$ which maximises the reward expressed by $R(F(\vec{X}, \vec{u}))$.

5.6.5 Evaluation of the Resulting World States

In the action exploration, an action taken by the robot is not always resulting in good states, but in most situations it is yielding states that are closer to the states with high reward. For this sake, a careful evaluation strategy is needed to be able to guide the learning system to a better achievement in the future.

In this applied learning mechanism, the quality of an action is measured through the resulting world state. The quality of the world state is quantified by using an evaluation function. Principally, the quality is measured by observing the changes of the relative position between the robot end-effector and the object resulting from the robot action, indicated by δ_{act}.

$$\delta_{act} = \Delta_{position}^{previous} - \Delta_{position}^{current} \tag{5.5}$$

where $\Delta_{position}$ at the previous and the current states represent the Euclidean distance between the robot end-effector and the target object. $\Delta_{position}$ is calculated as follows :

$$\Delta_{position} = \sqrt{\Delta x^2 + \Delta y^2} \tag{5.6}$$

where Δx and Δy is the object's relative position coordinates with respect to the robot end-effector. Positive signs of δ_{act} represent robot motions which are approaching the target object. On the contrary, negative signs represent robot motions leaving the target object.

Based on this value, the reward $R(t+1)$ representing the quality of the resulted world state is determined :

$$R(t+1) = \begin{cases} 0.0 & if \quad \delta_{act} \rightarrow \text{low quality} \\ 0.5 & if \quad \delta_{act} \rightarrow \text{medium quality} \\ 1.0 & if \quad \delta_{act} \rightarrow \text{high quality} \end{cases} \tag{5.7}$$

The quality of δ_{act} is specified depending on the implementation. If the robot does not neither approach the object at all nor leave the object, an intermediate reward value is produced. The intermediate reward value is useful to avoid too much noise to be applied when the action was only slightly worse. Otherwise, the action would deteriorate.

5.6.6 Noise Control Strategy

The noise generated by the noise adder is represented by a uniform distribution[1] of random numbers between - 1.0 and 1.0 representing the same range of the neural network internal values. The noise values are adjustable, based on the output of the prediction network which are real numbers between 0.0 and 1.0. The basic strategy for controlling the noise is described as follows :

$$\vec{n} = \begin{cases} 0.0 & if \quad R^*(t+1) \rightarrow \text{high} \\ \text{medium range random number} & if \quad R^*(t+1) \rightarrow \text{medium} \\ \text{high range random number} & if \quad R^*(t+1) \rightarrow \text{low} \end{cases} \tag{5.8}$$

where the exact values limiting each class of random number and $R^*(t+1)$ are determined further in the implementation.

The noise is controlled separately for every output, so that :

$$(\dot{\theta}_1, \dot{\theta}_2) = (\dot{\theta}_1^* + n_1, \dot{\theta}_2^* + n_2) \tag{5.9}$$

where $\dot{\theta}_1^*$ and $\dot{\theta}_2^*$ are the joint velocity values output by the action network.

5.6.7 Phases in the Execution

The entire system execution involving the above described components is done by carrying out two separate phases : an **action phase** and an **evaluation phase** sequentially from time to time.

[1] The selection of this form of distribution is based on the noise handling procedures described in [Werbos, 1990].

Action Phase

The action phase is the phase in which the robot actions are determined as depicted in fig. 5.9. In this phase, the action network generates its output $\vec{u}(t)$, in the form of joint velocity values $(\dot{\theta}_1^*, \dot{\theta}_2^*)$.

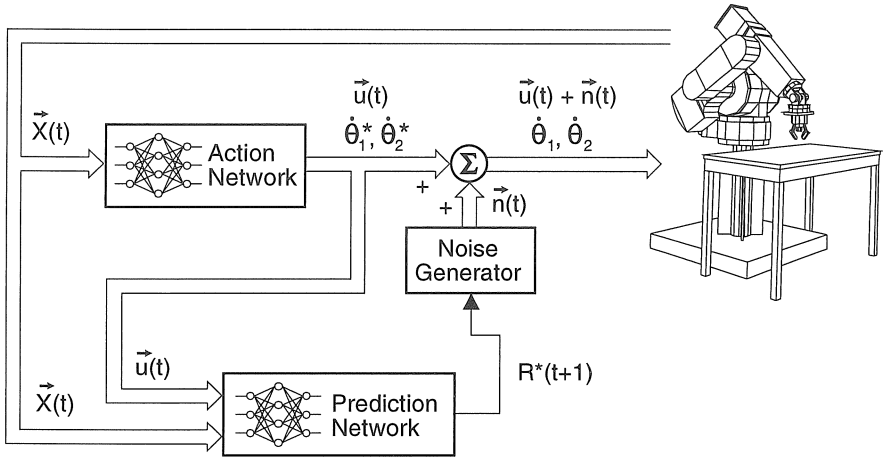


Figure 5.9: The prediction network performs the control of noise, based on its prediction of the quality of the action network output.

Based on the action network's current input-output pair $\vec{X}(t)$ and $\vec{u}(t)$, the trained prediction network predicts the next world state $\vec{X}(t+1)$ and produces the corresponding predicted state quality $R^*(t+1)$. This predicted value then bases the control of the noise generation. The new joint velocity values to which noise has been added $(\dot{\theta}_1, \dot{\theta}_2)$ are then sent to the robot.

By assuming a logical procedure to represent each process step, the action phase can be summarised as follows :

```
{
    forward_action_network();
    forward_prediction_network();
    add_noise(PREDICTION_NETWORK_OUTPUT);
    move_robot(JOINT1_VEL,JOINT2_VEL);
}
```

`forward_` represents the recall operation of a network and `JOINT1_VEL` and `JOINT2_VEL` are the velocity values of joints 1 and 2 sent to the robot arm.

Evaluation Phase

In the evaluation phase as depicted by fig. 5.10, the quality of the new world state $\vec{X}(t+1)$ is evaluated. The evaluation function outputs the judgement of this quality $R(t+1)$ which is used in the training of the prediction network.

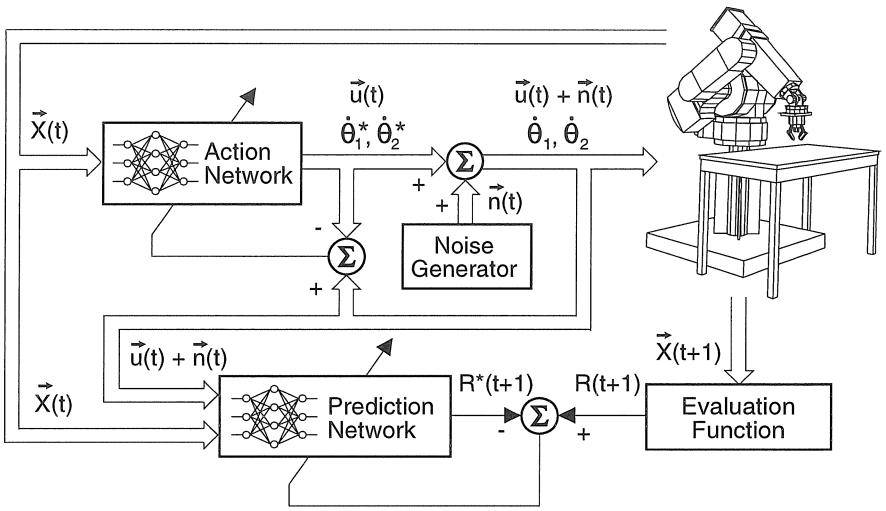


Figure 5.10: The prediction network is adapted based on the evaluation result of the current control signal.

The prediction network is trained by using the standard back-propagation algorithm. The input pattern for the training of the prediction network is :

$$P_i^{pred} = (\vec{X}(t), \vec{u}(t) + \vec{n}(t)) \tag{5.10}$$

and the output pattern is :

$$P_o^{pred} = (R(t+1)) \tag{5.11}$$

At the same time, if $R(t+1)$ is maximum, which implies that the system output $(\vec{u}(t) + \vec{n}(t))$ resulted in a "good" situation, the training of the action network is performed. This network is trained by using the input-output of the whole controller which generates that "good" situation, as follows :

$$P_i^{act} = (\vec{X}(t)) \quad (5.12)$$
$$P_o^{act} = (\vec{u}(t) + \vec{n}(t)) \quad (5.13)$$

The i and o subscripts indicate the input and output patterns consecutively. The standard backpropagation algorithm is used for the training of the action network as well.

By assuming a logical procedure to represent each process step, the evaluation phase can be summarised as follows :

```
{
   evaluate_action();
   train_prediction_network();
   if (reward == 1.0) train_action_network();
}
```

where `evaluate_action()` is carried out based on the specified evaluation function.

5.7 Aspects Influencing the System Behaviour

There are a number of issues that are significant in finding an optimal setting of the neural network controller scheme. Many of them are dealing with selection of parameters that have to be done experimentally. In this section, some important issues are discussed and verified by simulations[2].

5.7.1 Effects of Action Exploration Policy on the System Consistency

A common problem arising in the reinforcement based learning algorithms is the consistency of the system's actions. In the exploration process, the system's actions are decided stochastically, thus the probability of performing a learned action more than once is low.

The main components that affect the action consistency are the evaluation function and the noise control strategy.

[2] A dedicated simulator program is developed and described in appendix C.

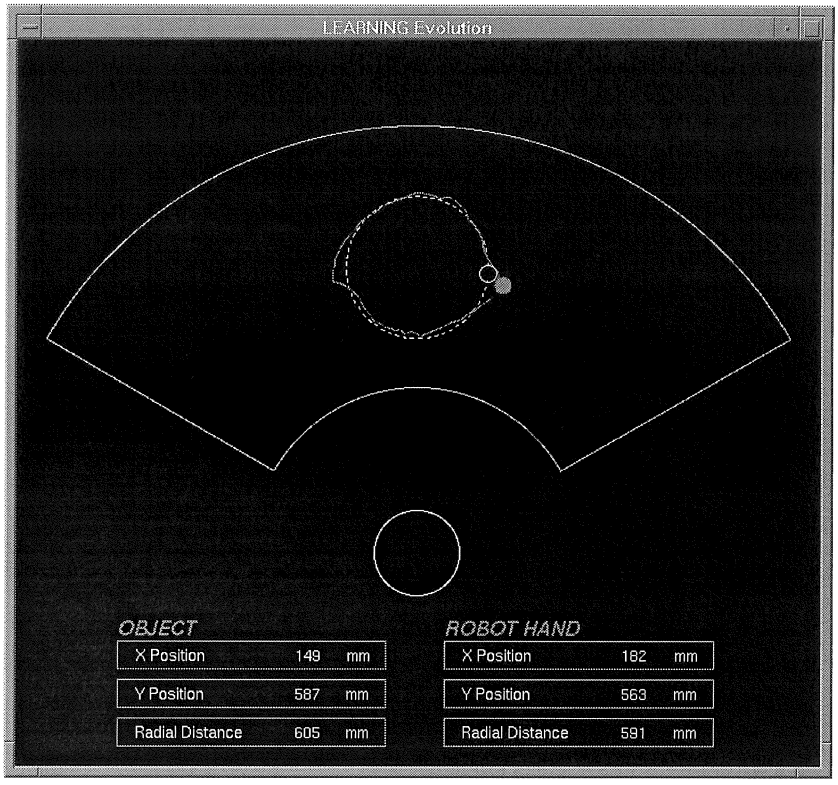


Figure 5.11: The simulator display of the robot 2-dimensional workspace represented by its top view, while performing an example of the tracking task. A fraction of the robot workspace is shown which is centered at the base of the robot arm (indicated by a circle lying outside the workspace). The object (indicated by a black circle) is moving performing a certain trajectory shape (a circle, printed in dotted line) while the robot end-effector (indicated by a grey circle) is tracking the object (with a grey lined tracking trajectory). Both end-effector and object locations are shown in the bottom panel of the simulator display. The radial distance is expressed with respect to the robot base.

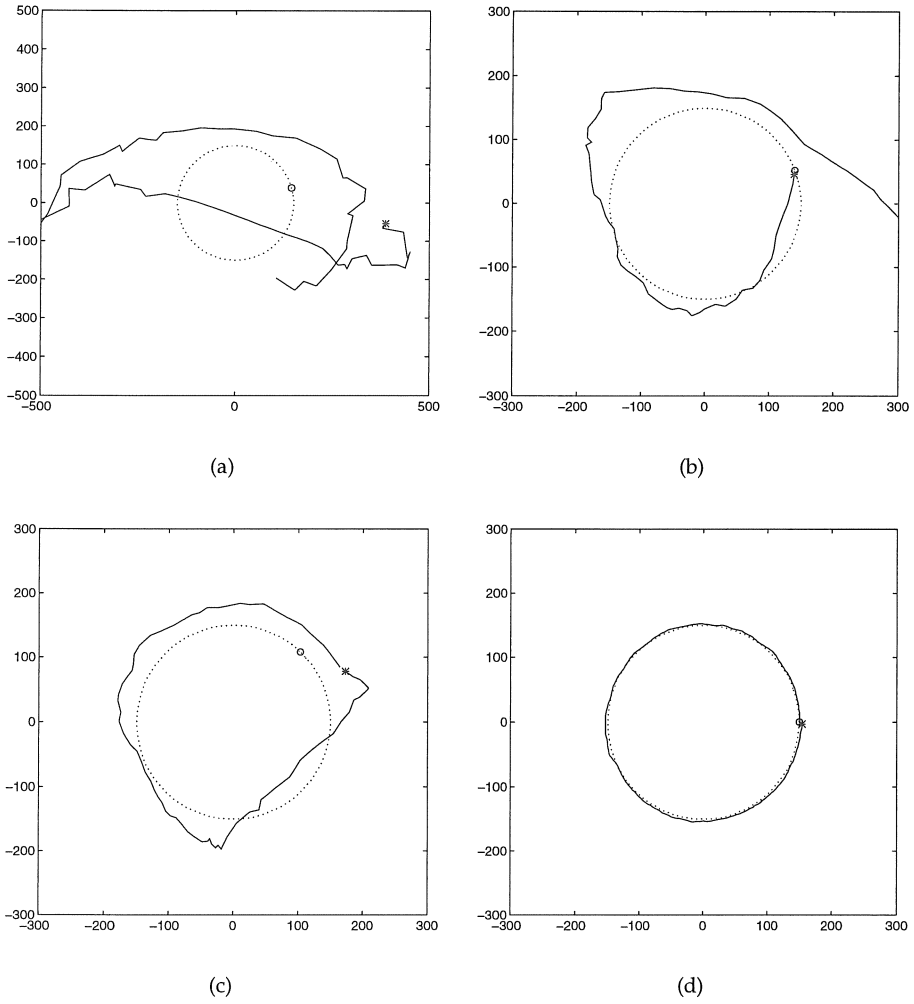


Figure 5.12: An example of the evolution of the tracking performance achieved during on-line learning. The object (represented by a 'o' mark) is moving performing a circular trajectory (printed in dotted line) in the counter clock-wise direction with a constant velocity. The object circulates continuously while the robot learns how to track it. The weights of action and prediction networks are initialised by random numbers. In every sampling interval, the object moves 1 step and the robot carries out 1 trial cycle performing 1 action step. One full object circular trajectory takes 70 steps. The performances are displayed at (a) 350 trial cycles, (b) 420 trial cycles, (c) 770 trial cycles and (d) 1260 trial cycles. The robot end-effector is represented by a '$*$' mark and its motion path is represented by a solid line.

Evaluation Function

The evaluation function expressed earlier by equation 5.7 is in practice implemented in the following form :

$$R(t+1) = \begin{cases} 0.0 & if \quad (\delta_{act} < -limit_{\delta}) \\ 0.5 & if \quad (-limit_{\delta} \leq \delta_{act} \leq 0.0) \\ 1.0 & if \quad (\delta_{act} > 0.0) \text{ or} \\ & \quad ((x_{eff} = x_{obj}) \text{ and } (y_{eff} = y_{obj})) \end{cases} \tag{5.14}$$

where x_{eff} and y_{eff} indicate the robot end-effector position and x_{obj} and y_{obj} indicate the object position. The value $limit_{\delta}$ is expressed in image pixel units. The highest evaluation value 1.0 is generated when the end-effector is in the move of approaching or has reached the target object. If an action causes the end-effector to leave the object for more than $limit_{\delta}$ pixels in the camera field, the evaluation function produces a 0.0 reward. If the robot end-effector does neither approach the object at all nor leave the object for more than $limit_{\delta}$ pixels, an intermediate critic value of 0.5 is produced.

By applying an experimentally chosen $limit_{\delta}$ value of 50.0, above form of evaluation function has led to a performance shown in fig. 5.11. The complete learning evolution towards that achieved performance is shown in fig. 5.12.

Effects of the Hidden Layer in the Prediction Network

The task of the prediction network is basically to predict what the evaluation function will output if a new world state has been achieved based on the current action execution. The effectiveness of the dynamic model of the environment built in the prediction network by learning is depending on how well the network can acquire and store the knowledge about the environment.

As a multilayer feedforward network can be used as an universal approximator [Hornik et al., 1989], the main influencing aspect that determines the network mapping capacity is the size of the network. Here, the determination of the optimal number of hidden neurons is carried out experimentally. Figure 5.13 shows the world state prediction accuracy achieved by four different sizes of prediction networks. The predicted action quality is compared with the real action quality quantified by using the evaluation function.

The important predictive performance required from the prediction network is the capability to distinguish the three different reward levels. The exact nominal output values of the predicted reward : 0.0, 0.5, or 1.0 are not necessary to be attained.

From the comparison presented in fig. 5.13, the network with 5 hidden neurons can be judged as the most optimal one. Although the prediction val-

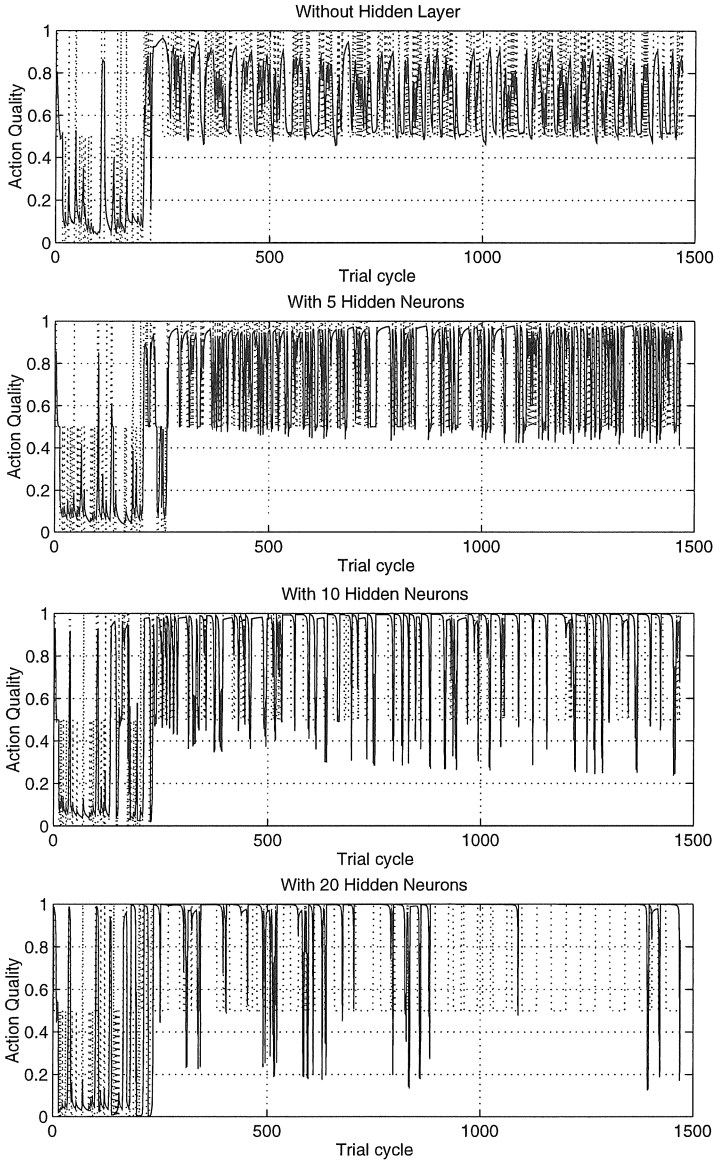


Figure 5.13: The accuracy of the dynamic model of the environment built in the prediction network. Dotted line indicates the measured current action quality and solid line indicates the predicted one. Four sizes of networks with different number of neurons in the single hidden layer are compared. This comparison is made during the first 1470 trial circles of tracking a target object moving with a circular trajectory. The neural network weights are initialised randomly.

ues are not exactly equal to any of 0.0, 0.5, 1.0, the responses to the 3 different reward classes are clear. In practice, the inexact prediction values can be handled by a thresholding operation.

Noise Control Strategy

The thresholding of the prediction network output is done in the noise generation procedure. Three noise boundaries are used (see equation 5.8), and are selected during executions based on the prediction of the next world state. Applying the prediction network with five hidden neurons, logically three world state threshold values must be chosen, as, together with their corresponding noise boundaries, are expressed in equation 5.15.

$$\vec{n} = \begin{cases} 0.0 & if\ (R^*(t+1) \leq 1.0) \\ -0.1 \leq r \leq 0.1 & if\ (R^*(t+1) \leq 0.9) \\ -0.5 \leq r \leq 0.5 & if\ (R^*(t+1) \leq 0.3) \end{cases} \tag{5.15}$$

where r is an uniformly distributed random number.

In fact, by being simply based on 3 fixed noise boundary values, an accurate, smooth end-effector motion cannot be obtained, as shown in fig. 5.15(a). The system is not able to generate a fine movement if $\Delta_{position}$ is small. As a matter of fact, robot motions have to be made finer around the object. Thus, the action exploration has to be sensitive to this condition.

To handle this requirement, another noise control strategy is applied with proportional noise values with respect to the distance between the end-effector and the object, expressed as follows :

$$\vec{n} = \begin{cases} 0.0 & if\ (R^*(t+1) \leq 1.0) \\ r \times (\Delta_{position}/c) & if\ (R^*(t+1) \leq 0.9) \\ -0.1 \leq r \leq 0.1 & if\ (R^*(t+1) \leq 0.5) \\ -0.5 \leq r \leq 0.5 & if\ (R^*(t+1) \leq 0.3) \end{cases} \tag{5.16}$$

where c is a constant which refers to the possible maximum range of the end-effector locations in the workspace.

The result of this improved strategy is shown in fig. 5.15(b) which is both smoother and more accurate. As shown in fig.5.14, the noise values are added more effectively to the joint velocity values. This strategy is also affecting the speed of the learning process, as shown in fig.5.14. The proportional noise boundary strategy can reduce the action exploration (indicated by the high noise values) earlier.

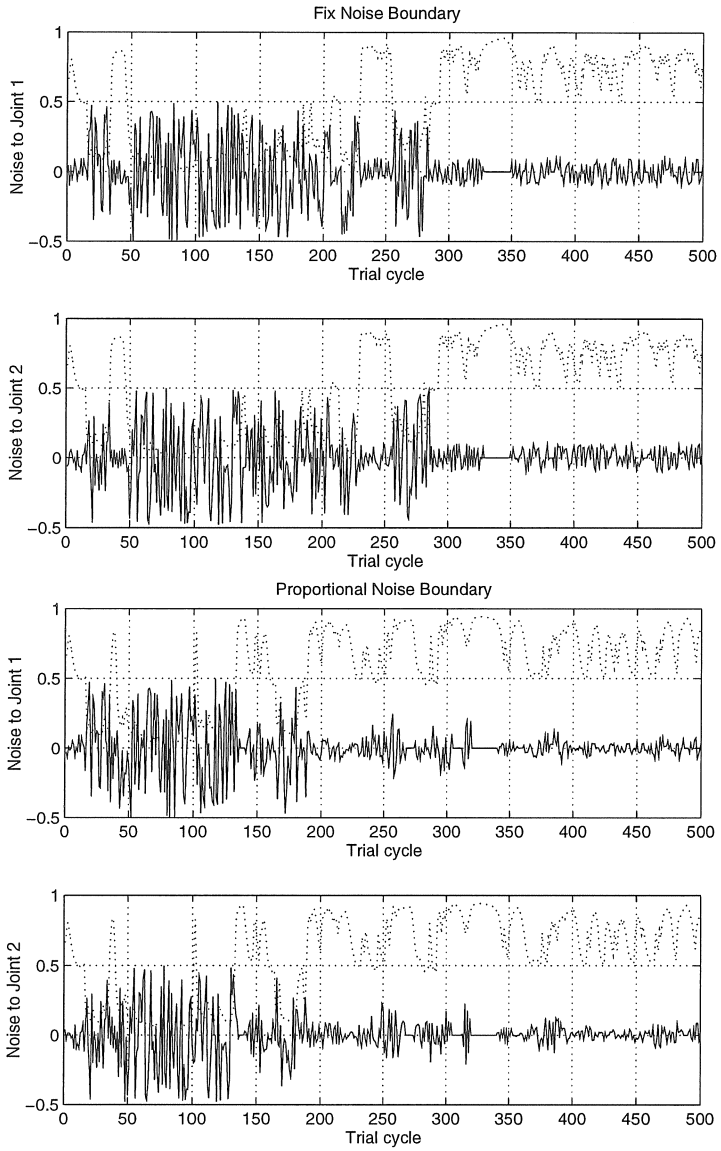


Figure 5.14: The noise values added to the robot joint velocity values, generated by using the fixed noise boundary and the proportional noise boundary strategies. The solid line indicates the noise values and the dotted line indicates the corresponding predicted next world state qualities.

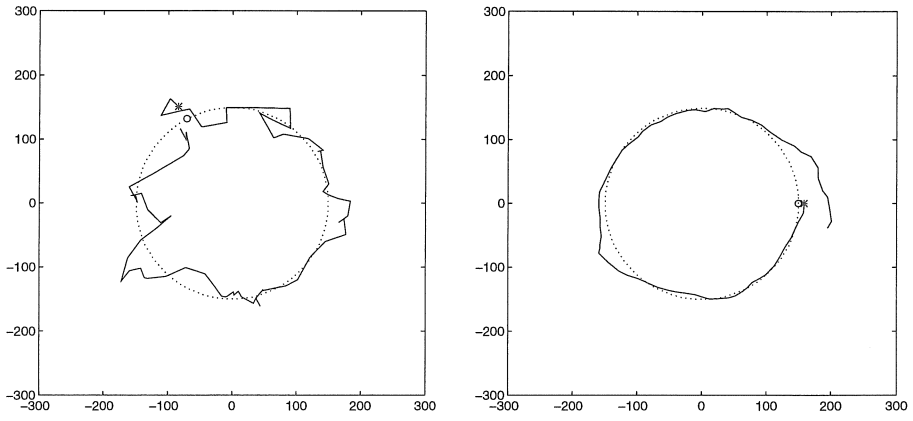


Figure 5.15: The tracking performance when the added noise are controlled by using (a) the fix noise boundary, and (b) the proportional noise boundary strategies.

5.7.2 Trajectory Prediction Using A Time-Delay Network

Particularly in the tracking control, the system capability to predict the near future target states may enhance the tracking performance [Katupitiya, 1985]. In the case of visual tracking of moving objects, the future states are represented predominantly by the future object's trajectory. In common prediction problems, the prediction of the next states is mainly performed by observing the history of that state. From a neural network viewpoint, this problem leads to the a problem of learning time sequences. There are actually three distinct tasks in the problem of learning sequences [Hertz et al., 1991] :

- **Sequence Recognition.** This task deals with the generation of a particular output pattern when (or perhaps just after) a specific input sequence is seen. There is no need to reproduce the input sequence.
- **Sequence Reproduction.** In this case the network must be able to generate the rest of a sequence itself when it sees part of it. This is the generalisation of auto-association (see section 3.5) or pattern completion to dynamic patterns.
- **Sequence Association.** In this general case a particular output sequence must be produced in response to a specific input sequence. The input

and output sequences might be quite different, so this is the generalisation of hetero-association (see section 3.5) to dynamic patterns. It includes as special cases pure sequence generation and the previous two cases.

In this visual tracking problem, the prediction task needed to be satisfied can be categorised as belonging to the sequence recognition problem and not to the sequence reproduction one. This is because, in each trial cycle, the system needs to produce only one single action.

Here, this state prediction problem is handled by applying a time-delay neural network [McClelland and Elman, 1986]. Time-delay neural networks perform sequence recognition by turning the temporal sequence into a spatial pattern at the input layer of a network. In this way, the conventional back-propagation methods can be used to learn and recognise sequences. The wide applications of time-delay neural networks can be found mostly in the field of speech recognition [Tank and Hopfield, 1987, Waibel, 1989].

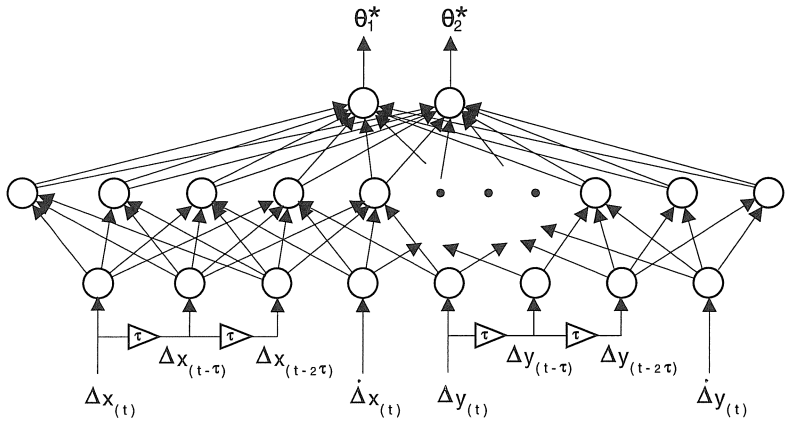


Figure 5.16: The topology of a time-delay neural network with 1 hidden layer applied as the action network. All input neurons are fully connected to the hidden neurons.

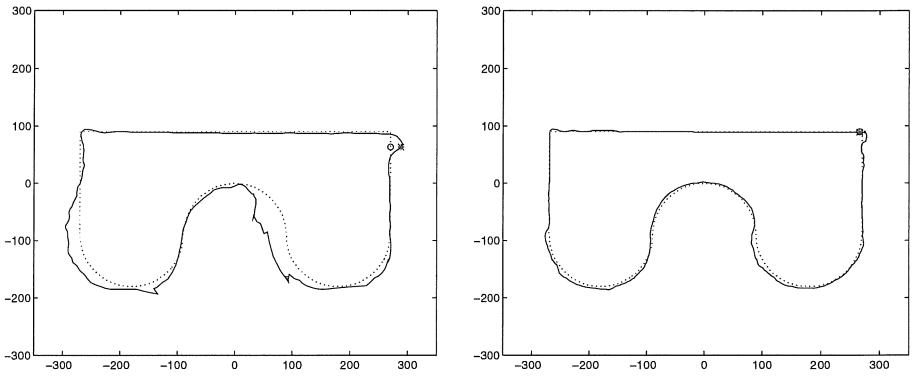
The turning of the temporal sequence into a spatial pattern is done by feeding the input signal into a *delay line* that is tapped at various intervals. For a given signal $\mathbf{x}(t)$ the delay line technique uses $\mathbf{x}(t), \mathbf{x}(t-\tau_1), \mathbf{x}(t-\tau_2), ..., \mathbf{x}(t-\tau_m)$ for the network inputs at time t, where m represents the delay order. In practice, a shift register can be used, in effect keeping several previous values in a buffer.

In the applied reinforcement learning scheme, a time-delay network is utilised as the action network. The delay line is applied to the object position state. A delay order m up to 2 is introduced. The time-delayed object position

state then can be expressed as $(\Delta x_{(t-\tau)}, \Delta y_{(t-\tau)})$, $(\Delta x_{(t-2\tau)}, \Delta y_{(t-2\tau)})$. Here, τ represents a fixed time-delay interval. In this way, the entire action network input P_i^{act} can be expressed by the following equation :

$$P_i^{act} = (\Delta x_{(t)}, \Delta y_{(t)}, \Delta x_{(t-\tau)}, \Delta y_{(t-\tau)}, \Delta x_{(t-2\tau)}, \Delta y_{(t-2\tau)}, \dot{\Delta x}_{(t)}, \dot{\Delta y}_{(t)}) \tag{5.17}$$

The topology of the time-delay action network is depicted in fig. 5.16. The performance improvement yielded by the implementation of the time-delay action network can be illustrated by the following figure. A more complex object trajectory is used to examine the improvement.



(a) (b)

Figure 5.17: The comparison of the system tracking performance on a more complex trajectory, achieved by utilising (a) a non time-delay action network, and (b) a time-delay action network. The comparison is done after 4000 trial cycles.

Obviously, the use of a time-delay neural network as the action network can reduce the tracking error. In can be seen that the robot tracks the object better than without using the time-delay action network. The system can track the sharp-cornered object trajectory more accurately and overshoots can be avoided.

5.7.3 Skill Extendability

The skill refinement process should be interpreted as a process to improve the action accuracy as well as a process to extend the system capability to over-

come unforeseen situations based on the gained experience. As the amount of experience is growing during continuous learning, the useful past experience should be maintained and combined with the experience gained more recently. Ideally the system must not forget the past experience if the more recent experience is collected. The latter is related to the skill extendability which should be considered as an important issue in the on-line learning problem. This section will examine the aspects on system skill extendability. Two variations are used as the observation focus : object trajectory shape and object velocity variations.

Responses to Variations in Trajectory Shape and Velocity

The evaluation of the system responses is focused on the system capability to overcome unforeseen states of object motion. To evaluate the response on changing object trajectory, different shapes of object trajectories are introduced, executed periodically for a certain interval. In this way, an evaluation about how well the system can maintain and combine the gained experience, can be performed. Two trajectories are used : a circularly shaped trajectory (as used in fig. 5.15), called trajectory 1, and a more complex shaped trajectory (as used in fig. 5.17), called trajectory 2.

The system response is observed in terms of system adaptation time needed to improve and maintain the tracking error level, right after the changes occur. Figure 5.18 presents the tracking error during a long tracking execution, in which the two trajectory shapes are introduced. The two trajectories are performed with the same velocity but in two opposite directions of motion (one clockwise and the other counter clockwise). The comparison is made with the same learning parameter settings, and the network weights are initiated with random numbers.

The same evaluation is done by varying the velocity of the moving object. The tracking error is presented in fig. 5.18, in which two values of object velocities are applied periodically, each as long as executing 3 complete trajectories. The second applied velocity value is 2 times higher than the first one. The more complex shaped trajectory is used.

As presented in the figure, the result shows that the learning system is able to adapt to changing object trajectories. During the first 6000 running cycles, the learning is clearly converging, minimising the tracking error. An interesting phenomenon occurs regarding the changes in the target object velocity. A system which has been trained for tracking at a higher object velocity, would be highly skillful for tracking at a lower object velocity, shown by a much lower tracking error values in fig. 5.18. On the contrary, a system need a certain period of time to adapt and learn to track at a higher object velocity if previously trained at a lower velocity level. However, in general the learning always converges.

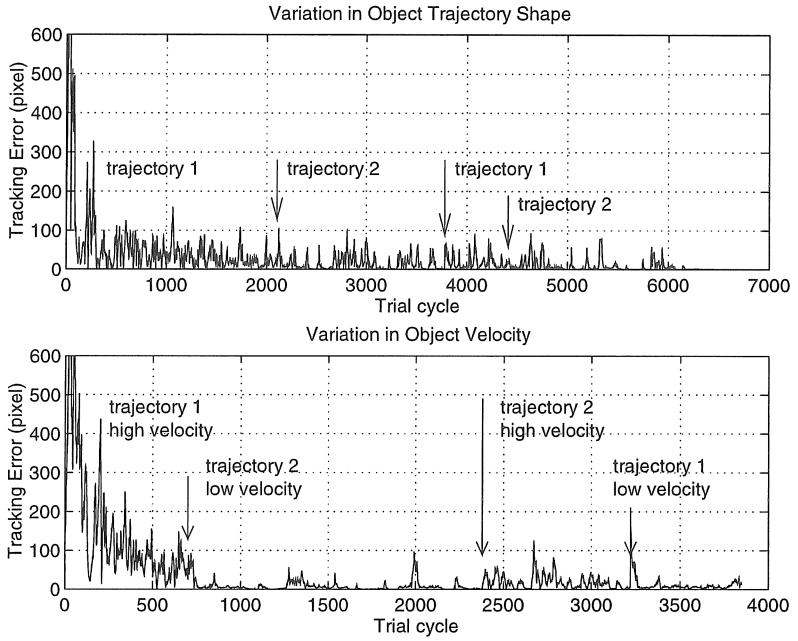


Figure 5.18: **Variation in object trajectory shape.** The tracking error recorded during a long tracking execution (started from random weight values) in which the object trajectory is changed periodically between a circular shaped trajectory (1) and a more complex shaped trajectory (2). Both are performed with the same velocity. Firstly, the object moves to make 10 turns of trajectory 1. Afterwards both trajectories are performed consecutively, each as long as 3 turns. Arrows indicate the starting time of the trajectories. — **Variation in object velocity.** The same running scenario is performed, in which here the object velocity is changed periodically between 2 velocity values. The higher velocity is 2 times higher than the lower one.

Effects of Hidden Layer in the Action Network

The system capability to store the knowledge of the learned state-action mappings is determined by the learning capacity of the action network. Since the complexity of the memory space of a feedforward network is determined by the number of its hidden neurons, the observation of the skill extendability should be focused firstly on this aspect.

The effect of the number of hidden neurons in the action network is presented in fig. 5.19, which records the tracking error during a set of task variations, with a similar running scenario with the one presented previously. An action network without hidden layer seems not capable of learning the state-action mappings. It can be seen that the tracking error is poorly reduced after

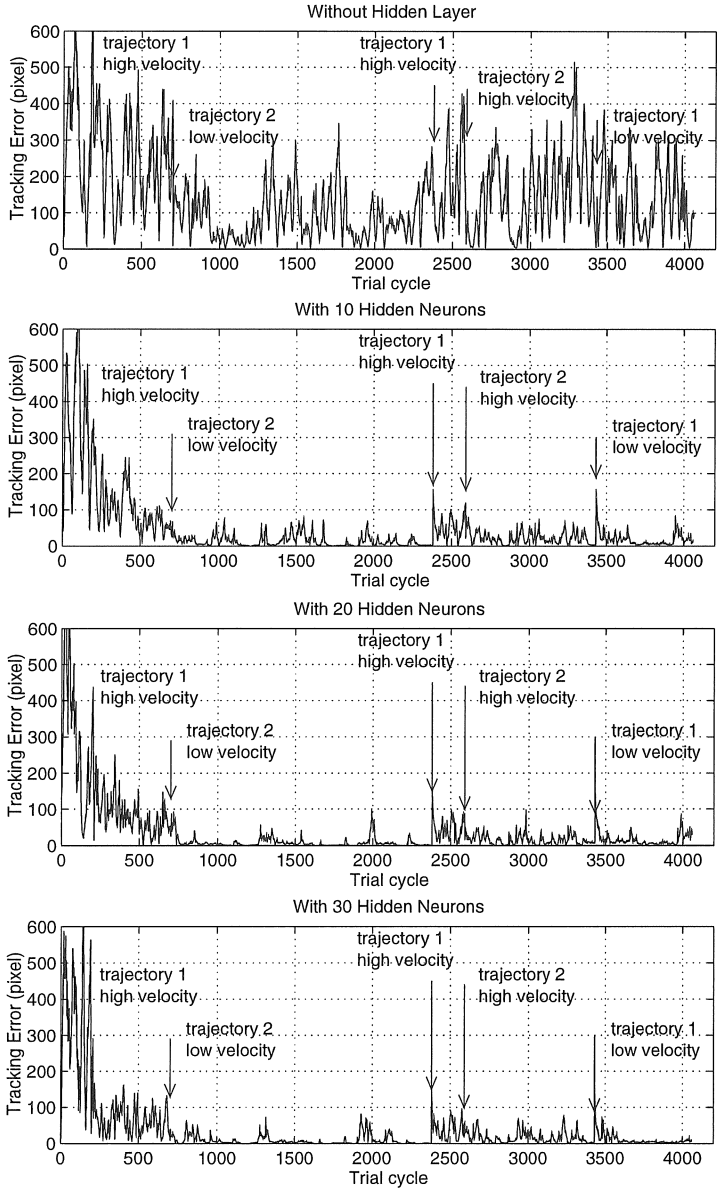


Figure 5.19: Learning performance as affected by the **number of hidden neurons** in the action network. The tracking error recorded during a long tracking execution (started from random weight values) in which the object trajectory shape and the object velocity is changed periodically. Firstly, the object moves to make 10 turns of trajectory 1. Afterwards both trajectories are performed consecutively, each as long as 3 turns. Arrows indicate the starting time of the trajectories. Two velocity values are applied. The higher velocity is 2 times higher than the lower one. The two trajectory shapes are performed in two opposite directions (clockwise and counter clockwise).

4000 trial cycles. On the contrary, networks with hidden layer obviously show their capability to learn the mapping. In general, action networks with hidden layers makes the system skillful to overcome the change of target object trajectories. But, in case of overcoming a suddenly increased object velocity, the system needs a certain time to learn and adapt to the higher motion velocity.

From this comparison, it is observed that the higher the number of the hidden neurons, the faster the tracking error reduced. This phenomenon particularly occurs within the first 700 cycles, where the network start to acquire the knowledge from scratch (the networks are initialised by random numbers). The average tracking error is also reduced during the next running as the number of the hidden neuron increases. In the case of more than 20 hidden neurons, the effect on reducing the tracking error is not so significant.

One reason to limit the maximum number of hidden neurons may be due to the system generalisation property. In general, a trade off has to be made between developing a generalised and a specialised network. If the system is too generalised, an accurate action will not be achieved. On the contrary, if the network is too specialised, the network may not be able to maintain the low tracking error when encountering a new situation. In the standard backpropagation, too specialised networks can occur if the number of hidden neurons is high, which implies that the network is too powerful. Since in this algorithm, the action networks is trained by pairs of input-output that probably appears only once and not by a fixed set of training examples, the chance to be overfitted is less.

A more reasonable consideration to choose the optimal maximum number of hidden neurons may be based on the required network recall time. It is clear that the higher the number of the applied hidden neurons, the larger the required recall time will be.

Effects of Action Network's Learning Rate

Learning rate plays its role in determining the step size towards convergence. The highest appropriate value of learning rate is constrained by the learning stability. A learning rate that is too large leads to unstable learning. On the contrary, a learning rate that is too small results in very long training times. In the case of using a fixed learning rate, its optimal value must be selected to compromise both conditions.

In this implementation, an optimal learning rate for the action network is chosen from experiments. An evaluation is done to observe the effect of different learning rates on the system skill extendability property. By using the same running scenario as used in the observation of hidden layer effects, a clear comparison is made on the aspects of both stability and capability to overcome new situations, where the result is presented in fig. 5.19

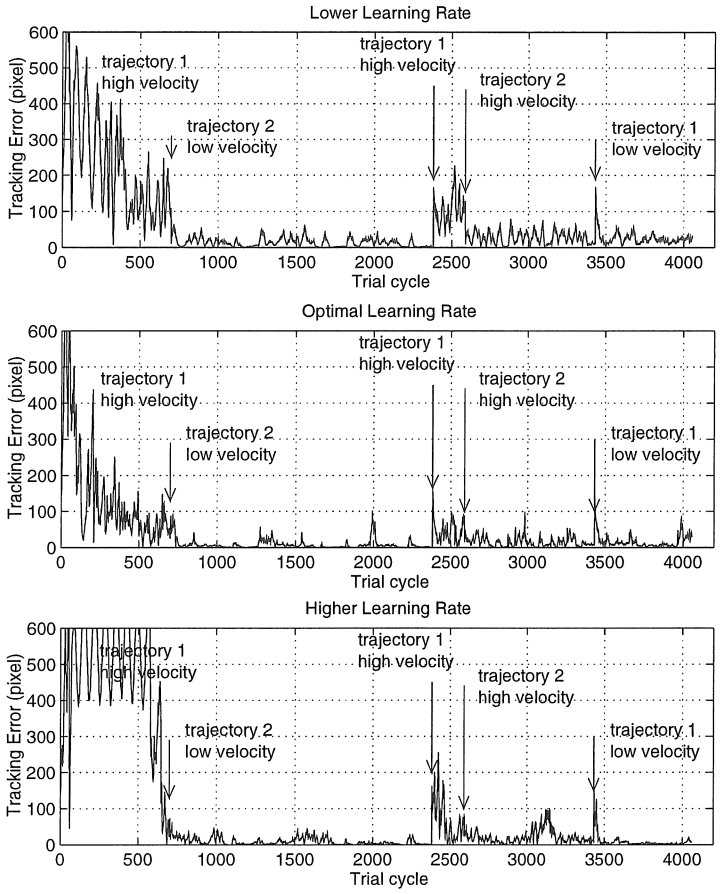


Figure 5.20: Learning performance as affected by the applied **learning rate** of the action network. The tracking error recorded during a long tracking execution (started from random weight values) in which the object trajectory shape and the object velocity are changed periodically. Firstly, the object moves to make 10 turns of trajectory 1. Afterwards both trajectories are performed consecutively, each as long as 3 turns. Arrows indicate the starting time of the trajectories. Two velocity values are applied. The higher velocity is 2 times higher than the lower one. The two trajectory shapes are performed in two opposite directions (clockwise and counter clockwise).

The system applying a learning rate lower than the optimal value, clearly converges more slowly, and a learning rate higher than the optimal one introduces a sort of system instability in the beginning of the learning process.

In overcoming an unforeseen higher object velocity, both systems with non-optimal learning values are showing a "shocking period" at the time when a velocity change occurs. At that instant, those systems do not have enough skill to learn from the past to act properly in the new situation. The system with the lower learning rate, could not use effectively the time it had in the past to learn from all situations it has encountered. On the other hand, the system with the higher learning rate, learns every new situation quicker and as a consequence, it forgets the past experience quickly as well. In general, those systems could not effectively utilise the previous situation they have encountered as a useful experience.

5.8 Discussion on the Learning Characteristic

One main fact that affects the learning characteristic on the applied method is that both action network and prediction network have to converge simultaneously to the appropriate solution. In this case, if the learning rates are not set appropriately, the system can converge to a state in which the prediction network decides that all input states will have a very poor expected performance. In such situation, the action network weights will not be updated and the system gets stuck.

Since the two networks are updated by pairs of input-output values, in which each pair probably appears only once, this method is somewhat less robust than the more standard version of backpropagation that learns from a set of prepared training examples. In addition, the signal that controls the learning of the action network – the prediction value – generated by the prediction network is not always correct, particularly when the prediction network has not converged yet.

Due to the way of network updating which is not based on a fix set of input-output pairs, here, the use of momentum factor α (see appendix A) will not be so effective.

In general, the reinforcement-comparison learning method is theoretically able to learn very complex functions, but tends to require many trial cycles before it converges. The time and space complexity for this algorithm O can be estimated as $O = M \cdot H$, where M is the number of input bits and H is the number of hidden neurons [Kaelbling, 1993].

5.9 Implementation on a Real Robot System

To verify the applicability of the neural network visual tracking controller, an implementation on a real robot is done. Components of this real-life learning implementation are presented in the following sections.

5.9.1 Robot Arm System

An articulated 6-DOF industrial robot arm[3] – the real version of the simulated robot used in the simulation – as shown in fig. 5.21 is used.

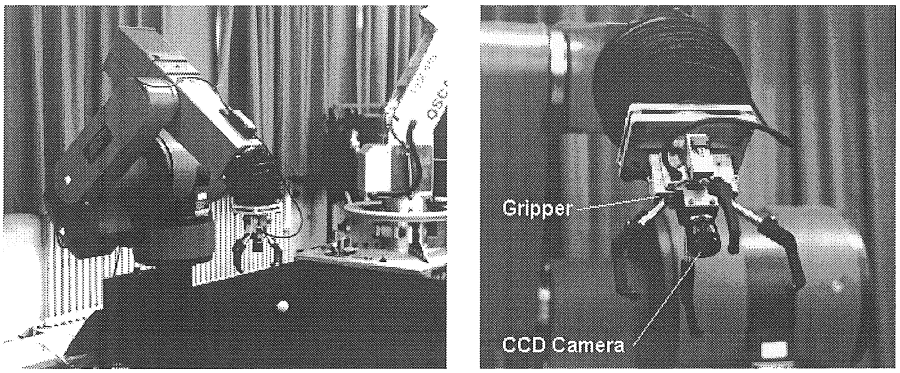


Figure 5.21: The robot arm system (left) and the camera mounted into the robot gripper (right) used in the implementation.

A CCD camera is mounted in the centre of a pneumatic-driven four-fingered gripper. As a simplified visual environment, a bright-coloured moving object is set on a dark-surfaced horizontal plane.

The description of the robot kinematic parameters and the velocity relationships among all joints for the specified task are presented in appendix D.

5.9.2 Control Hardware

Figure 5.22 describes the transputer system used in the implementation. Four transputers are interconnected. Their connections are established through either transputer links or VME buses. The development environment, which is a multi-user environment, is based on a UNIX system.

[3] A commercial industrial robot : American Robot™ type AR 6500.

The transputer based vision system GM8104 (transputer 2 on figure 5.22) is used to handle the frame grabbing task. This system is a video resolution 32 bit graphics system, which is supported by a T800 type transputer with 1 Mbyte of 100ns local DRAM. A B/W CCD camera and a monitor are connected to this transputer. The video memory consists of 4 Mbytes of dual ported video ram (VRAM). This is sufficient memory for 2 screens of 768 $\times$ 576 $\times$ 32 bits. The 32 bits per pixel are divided into 8 bits Red, 8 bits Green, 8 bits Blue and 8 bits Alpha.

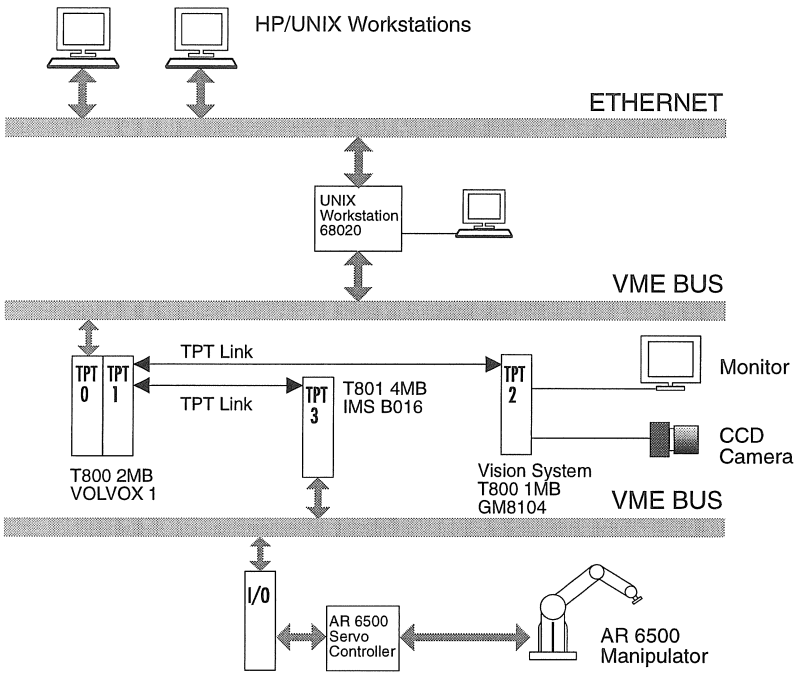


Figure 5.22: The lay-out of the transputer system used for the overall control implementation.

To perform the real-time object localisation task, both linked transputers 1 and 2 are used. The computing load of the calculation is shared between those transputers. The detail of the implementation of this real-time object localisation technique is described in appendix E. Transputer 3 is mainly dedicated to the control of the AR 6500 robot system. It is connected to an I/O board through VME bus. This I/O board manages the communication of the controlling signal from transputer 3 to the robot servo controller and the current

joint position information on the opposite way.

5.9.3 Neural Network Controller Implementation

The neural network controller is incorporated into the joint velocity control mode of the robot[4] (running on transputer 3). The overall control scheme is shown in fig.5.23.

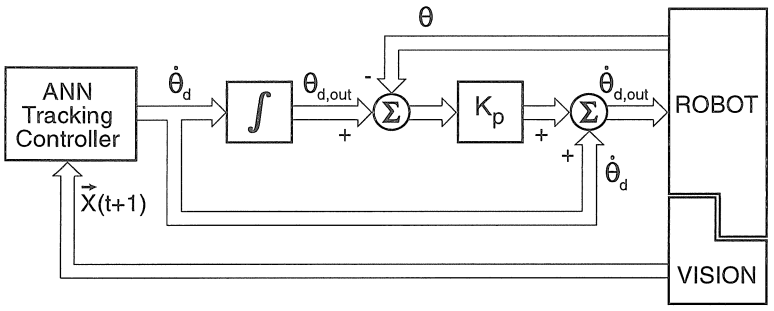


Figure 5.23: The incorporation of the neural network controller into the robot joint velocity controller mode.

The desired joint velocities $\dot{\theta}_d$ output by the neural network controller are integrated to desired joint positions $\theta_{d,out}$. A position feedback loop with feedback gains K_p (units s^{-1}) is added inside the external control system resulting in output velocities $\dot{\theta}_{d,out}$. A visual feedback loop conveying the new world state $\vec{X}(t+1)$ is carried out from the vision system to the neural network controller. The joint controller loop works at 150 Hz; the visual feedback and the neural network controller work at 5 Hz.

5.9.4 Performance Evaluation

The evaluation of the learning performance on the real robot system is mainly emphasising on the skill gradual improvement. Two issues are particularly observed : the system reflexivity and the tracking accuracy. The evaluation is done by comparing relative system performances at several different trial cycles. In this way, the evaluation of the learning phenomena can be done independent from the robot maximum speed nor from the sampling rate of the vision system. In this implementation, the following system configuration is applied :

[4]The neural network controller is self-developed by using C-language and the joint velocity controller is implemented by an in-house developed robot controller software package called COMRADE [Van de Poel et al., 1993].

- 16 hidden neurons in the time-delay action network, with the optimal learning rate
- no hidden neuron in the prediction network
- the proportional noise boundary.

Prediction network with no hidden layer is applied for the sake of shorter total execution time.

System Reflexivity

The achievement of a system learning reflexive behaviour should be first evaluated in the terms of its reflexivity. An experiment is set up to evaluate this reflexivity through an evaluation of the system step response. This experiment is done by introducing to the system an step stimulus in the form of a static object to be reached that appears in the camera field suddenly. The set-up is shown in fig. 5.24. A light source is used to allow the sudden appearance of a target object. The target object stays statically until the robot can reach it.

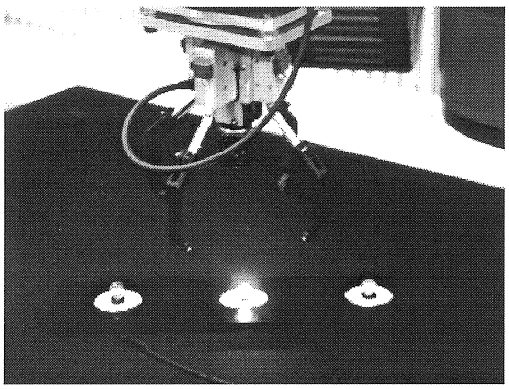


Figure 5.24: The experimental set-up to evaluate the system response to a step stimulus. A series of lights with a distance of 130 mm located statically in the robot workspace is used.

The robot step responses at different trial cycles are shown in fig.5.25. It is shown clearly that the reflexive behaviour is improved with respect to the number of system trial cycles.

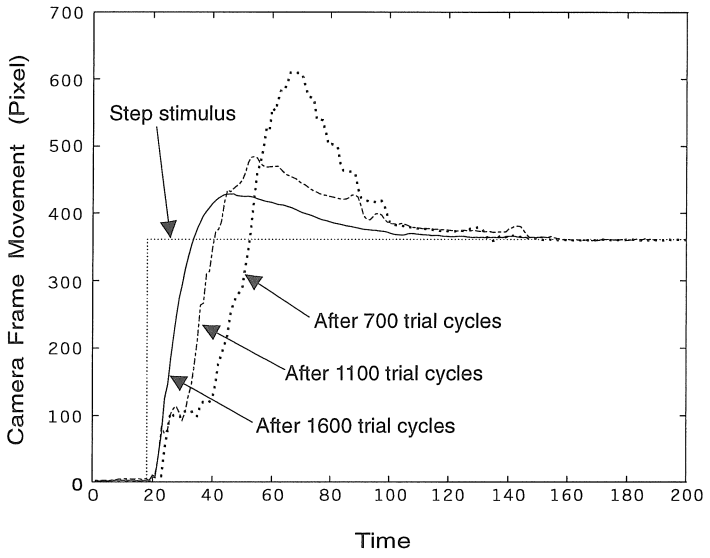


Figure 5.25: The step responses achieved by the system after different numbers of trial cycles. This data is sampled at 5 Hz.

Tracking Accuracy

The next significant issue to be evaluated is the system tracking accuracy. This issue is crucial in terms of real time performance. Besides the quality of the learned behaviour, the tracking accuracy also strongly depends on the maximum motion speed that the robot arm can achieve and the sampling rate of the visual information. By comparing the performance at different trial cycles in the same tracking task set-up, an evaluation of the skill gradual improvement is done.

An experiment is set for tracking a target object moving on a circular trajectory. The object moves with a constant linear velocity of about 10 mm/s. Figure 5.26 shows the gradual improvement of the achieved accuracy.

As presented in the figure, the system is not able to track the object yet after 1900 trial cycles. After about 3000 trial cycles, the system begins to keep on tracking the moving object. The more trial cycles carried out afterwards the better becomes the tracking accuracy, for instance as shown after 4000 trial cycles. The improved accuracy is presented by the smaller error variation range observed in the camera field. The tracking error expressed in the camera field axes is presented in fig. 5.27.

As an illustration for the robot tracking skill, fig. 5.28 shows the learning robot when tracking a moving toy train.

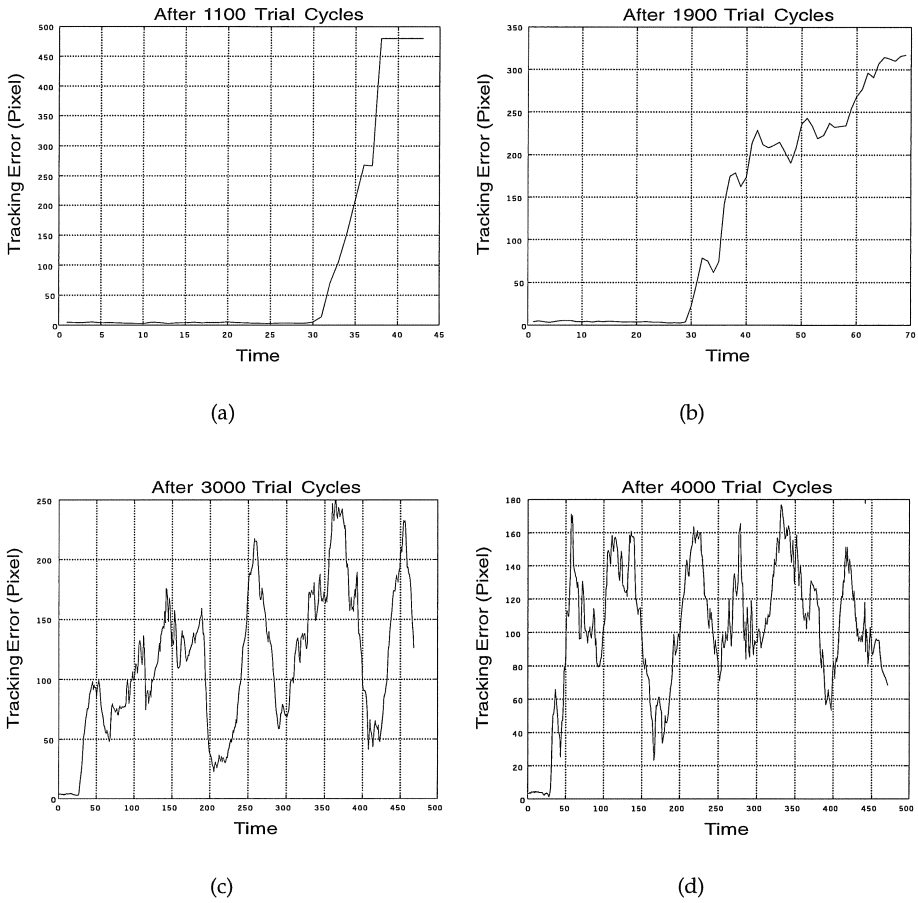


Figure 5.26: Tracking performance achieved by the system after different numbers of trial cycles. The above data represents the system responses, starting from a time when the object is standing still. After a certain instant (about 30 time units), the object is moving performing a circular trajectory on a constant linear velocity of about 10 mm/s. The system is not able to track the object yet after 1900 trial cycles, shown in (a) and (b) where the object is running away from the camera field. In (a), the system reacts very poorly and immediately looses the object. In (b), for a certain period in the beginning, the system is able to track the object before the object is running away from the camera field. Data in (c) and (d) present the recorded tracking error during 2 turns of moving on the circular trajectory.

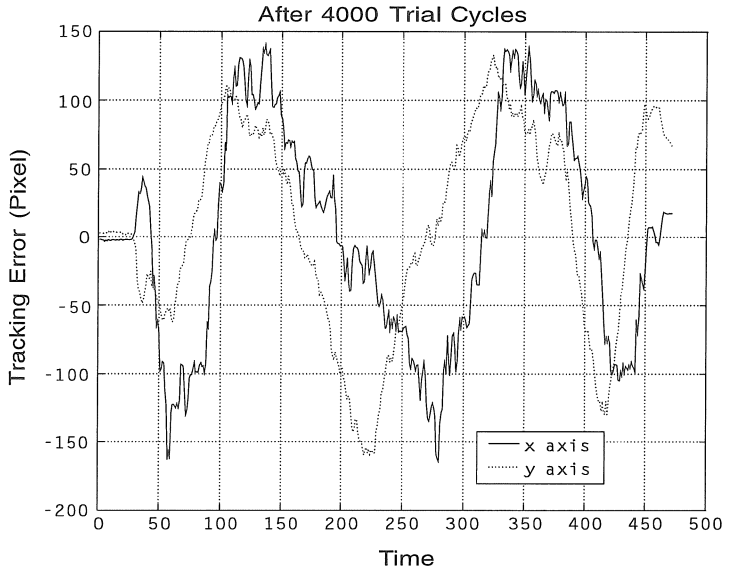


Figure 5.27: The tracking error of the same task as presented in fig. 5.26(d), expressed in the camera frame axes.

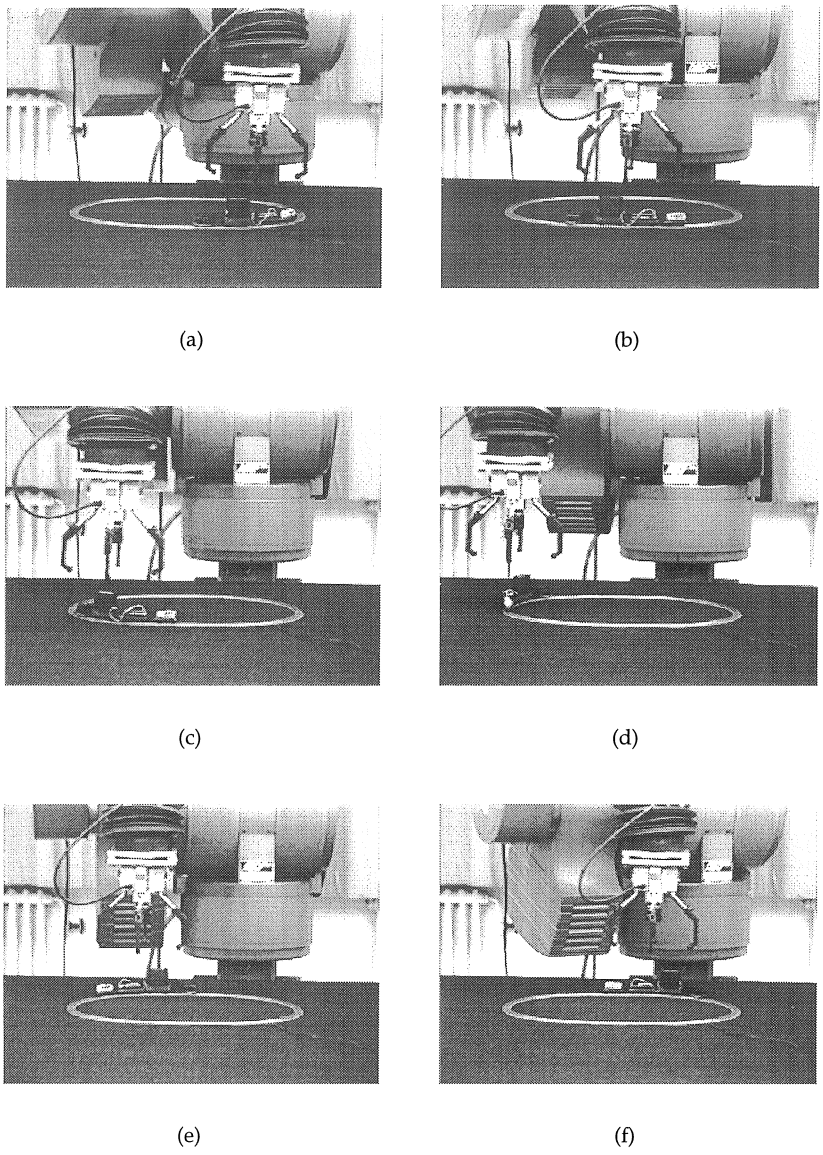
5.10 Conclusions

The main conclusion to be drawn from this chapter is that skill refinement of a robot's reflexive behaviour can be performed through learning, based on a reinforcement mechanism.

Reinforcement-comparison learning algorithms can be used to perform a gradual improvement of the robot visual tracking behaviour, based on an immediate reward. The results are encouraging but also indicate that the applied learning approach involves a number of key aspects. More specifically, this learning approach implies some requirements :

Good exploration strategy. Since the action mapping is learned through searching stochastically the appropriate action for every situation, the effectiveness of the exploration strategy is very crucial. This effectiveness will determine the required learning time as well as the quality of the learned actions. The main factor related to this issue is the accuracy of the dynamic model of the environment built-in in the reward predictor network. This is practically determined by the number of hidden neurons applied in the network.

Suitable immediate rewards for the resulting reflexive actions. Since the learning



(a) (b) (c) (d) (e) (f)

Figure 5.28: Sequential snapshots of the robot while performing an example task : tracking a battery-powered light carried by a moving toy train.

of the action network is controlled based on the reward values derived from the recently resulting situations, the representativeness of the reward values is very determinant. Unsuitable reward values can make the learning system get confused and the learning convergence becomes slower. The influencing factor for this aspect is the representativeness of the evaluation function.

Enough capacity for extending the skill. One supporting condition to establish skill refinement is the system capacity to extend the learned skill. This factor is influenced simultaneously by the number of hidden neurons and the learning rate applied in the action network.

System ability to predict object trajectory. Tracking performance increases when the system can predict the the near future object trajectory. This requirement can be fulfilled by applying a time-delay network as the action network.

Chapter 6

Learning Cooperative Motion Behaviour

Since emotions are few and reasons are many (said the robot, Giskard),
the behaviour of a crowd can be more easily predicted
than the behaviour of one person can.

ISAAC ASIMOV

6.1 Introduction

Many examples of coordination and cooperation can be found in real life. Not only when a group of people carry out some tasks together, but even in the action of each person, coordination and cooperation can be observed – for example when someone holds or manipulates objects using his or her fingers and arms dextrously.

In general, the concept of coordination can be referred to as working together to increase effectiveness. On the other hand, cooperation can be referred to as working together effectively for shared purposes. From a robotic point of view in particular, cooperation can be meant as a process of taking different observations and informations from the *same world* (the world includes the agents) and combining them in such a way that the process achieves a *common task* [Bajcsy, 1993]. Further is also stated in [Bajcsy, 1993] that a cooperation requires agents to collectively distribute *task-related actions* amongst

themselves in order to achieve the *top level task*. Another perspective from [Van Brussel, 1994] paraphrases that the flexible behaviour in the cooperation among several subsystems can be achieved when the subsystems can behave in a completely autonomous way, but try to combine their efforts to achieve an overall system goal.

A multi-robotic system is understood as a system which consists of several autonomous/semiautonomous robots working together trying to achieve a common goal [Taipale, 1993]. This system has been acknowledged as giving various advantages in the manipulation of complex tasks that cannot be performed satisfactorily by a single robot systems. Current research on multi-robotic systems can be viewed as concentrating on two main interests. One is on the implementation of two-arm robotic systems – which mostly puts emphasis on the investigation of the underlying control strategy and the optimisation of the motion trajectory, and the other is on the implementation of a group of more than two cooperative robots – which mainly puts emphasis on the development of the system's behaviour and flexibility.

Each system brings its typical advantages. In two-arm robotic systems, the advantages are mainly in the form of system capabilities to share the task operations and the load of the manipulated object (in terms of size and weight). Advantages are also obtained by utilising the two-arm redundancy to increase system's dexterity [Nakamura, 1991], and by taking benefit of diverse capabilities of different robots [Fukuda and Ueyama, 1994]. In a system involving more than two robotic agents, the main advantage is cooperation which enables a group of simpler robots to achieve the same goals as one big, complex robot. Furthermore, this multiplicity of robots can provide greater error sustainability because failure of one or more robots is not fatal for the overall system performance [Fukuda and Ueyama, 1994, Taipale, 1993].

As a behaviour, cooperation can be represented in several ways. In a broader sense, cooperation can be considered as a kind of natural behaviour that occurs in many social systems. In such systems, although the system's goal is able to be recognised or clearly defined, the behaviour itself is developed naturally through a gradual process of evolution aiming at the goal. In this natural case, the behaviour is certainly not built by going through a specific design phase. In any form, learning is often involved.

The human case of coordinating the two arms is an interesting example of learning cooperative behaviour. For sure, human beings coordinate their arms without knowing explicitly their arms' representation, e.g. arms' kinematic or dynamic models. This behaviour is built through learning from accumulated experience from their early age.

The work described in this chapter deals with building a coordinated control skill of two robot arms through neural learning. One of the tasks that human beings use to perform when coordinating their two arms simultane-

ously is to pose one arm by referring to the posture of the other arm when holding and manipulating an object. An analogical task, a task to pose the second robot arm according to the posture of the first robot arm, is chosen as a case study.

6.2 Cooperative Action of Multi-Robotic Agents

Multi-robotic systems have been investigated from different points of view, and various approaches have been used. So far, two streams of approaches have characterised this research field. Those are **model based control approaches** and **behaviour based control approaches**. Each stream has its own goal and emphasis.

6.2.1 Model Based Control Approaches

Model based approaches are mostly used in the control of two-arm robot systems. Approaches based fully on the kinematic computations are used mainly under consideration that the grasped object is rigid and there is no position error in the system during manipulation. As examples, in [Zheng and Luh, 1985, Lim and Chyung, 1987], methods based on closed kinematic chain analysis are proposed.

The more reliable control methods should also consider the dynamical aspect of the arms' interaction. In [Ishida, 1977] a method to incorporate the interaction force between two arms, obtained from a wrist-mounted force/torque sensor, has been proposed. This method decomposes the objects' motion into rotational and translational motions, and performs complex object motions by combining them, while the joint motor torque control mode is applied instead of the joint position mode. Another method implementing a hybrid position/force control mode is demonstrated in [Hayati, 1986]. This method derives the motion equations in a constrained coordinate frame located at the grasped object. The magnitude of force and torque in the object is minimised by means of the inverse kinematic model of each arms, which allow the computation of the force and torque values contributed by each arm. Another important research topic in this domain is in the two-arm redundancy problem, as clearly explained in [Nakamura, 1991]. A control strategy to utilise the relative motion between two arms in order to increase dexterity is studied in [Arthaya, 1995]. In this research, a new task specification is designed to allow two robot arms to perform a cooperative task and to exploit the redundant properties of the system. The coordinated motion between the two arms is realised by feedforwarding the velocity trajectories of the first robot to the second robot, or vice versa.

A lot of other methods have been proposed, mostly as improvements of earlier proposed, more generic methods. In general, model-based methods mainly handle tasks that can be explicitly synthesized in advance. In this case, the better the model that can be made, the more accurate the task execution that can be achieved.

6.2.2 Behaviour-Based Control Approaches

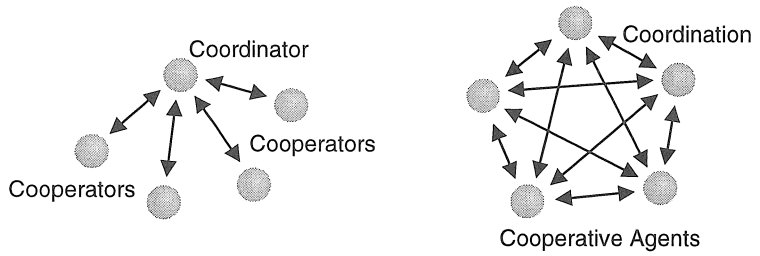
Behaviour-based control methods applied to multi-robotic system have been proposed by a number researchers [Fukuda and Ueyama, 1994, Asama, 1995, Taipale, 1993, Van Brussel, 1994], aiming at the realisation of flexibility, robustness and fault tolerance of the resulting system.

A system called **Cellular Robotic System (CEBOT)** is a recent research project directed toward studying the flexibility of multiple robotic systems [Fukuda and Ueyama, 1994]. CEBOT is a distributed robotic system consisting of functional units called *cells*. This system is a self-organising system, whose cells are autonomous and the whole system is organised autonomously and distributively. In CEBOT, intelligence is provided to each cell and the overall configuration is organised by the interaction among cells, which leads to the emergence of the whole system behaviour. The system's intelligent behaviour is not predefined in the system. Learning is one of the ways used to develop the individual cell behaviour as well as the multiple cells, in order to build mutual understanding among cells that contributes to the whole system behaviour.

Another recent research project addresses a concept called **holonic behaviour** [Van Brussel, 1994]. This concept brings a new vision about the cooperative control structures, which are quite different from the pure hierarchical construction. Although a hierarchy still exists, the organisational structure is much looser. The system consists of a set of *"holons"*. **Holon** is a synthetic word consisting of **holos** (total, whole) and **-on** (like in "neutron", referring to an individual body). Holons are simultaneously part and whole; they are part with respect to the higher levels in the hierarchy, but whole with respect to the lower levels. A system consisting of several subsystems exhibits holonic behaviour, when the subsystems (can) behave in a completely autonomous way, but (try to) combine their efforts to achieve an overall system goal. The main features of holonic systems can be derived from observing the behaviour of human systems and societies. A human holon simultaneously has two fundamental, rather opposite, behaviours namely : a **self-assertive tendency**, expressed by the holon's degree of **autonomy** to execute a given task; and an **integrative tendency**, expressed by properties like **goal-orientation, self-organisation, flexibility**, and **extendability**.

6.2.3 Multi-Agents Cooperation Types

When two or more autonomous robots work together at tasks, the configuration of multi-agent system can be classified into two types as shown in fig. 6.1 [Shibata and Fukuda, 1993].



(a) centralised system (b) decentralised system

Figure 6.1: Configurations of multi-agent systems.

One is a centralised system in which one *coordinator* or *master robot*, exists and other robots, *cooperator* or *slave robots*, obey the coordinator. The other is a decentralised system in which each robot performs tasks cooperatively. In the centralised system, the master robot is required to have many functions. If the coordinator robot breaks down, the task cannot be achieved. On the other hand, in the decentralised system, each robot has a function. Since plural robots work cooperatively, it is easy to deal with various tasks. Moreover, if one robot breaks down, other robots can help or replace the broken robot. Therefore, the decentralised system is superior to the centralised one. However, it is difficult to achieve the decentralised system, because interaction among robots influences the running of the entire system unfavourably when public resources (e.g. energy supply, tools, raw materials) are limited. Conflicts among robots might occur when using a public resource. The conflicts may cause collisions and deadlock states among robots. In order to avoid conflicts, it is necessary for the robots to communicate.

6.3 Coordinator – Cooperator Type of Cooperation : A Case Study

In daily activities, humans unconsciously perform a number of actions that involve the coordination of their arms. Many of them are even done without paying much attention about how the tasks are synthesized.

One of the approaches that human beings use when holding and moving an object by two arms is to let one arm (the first arm) hold and move the object more predominantly, while the other arm (the second arm) holds the object and follows the motion of the first arm. In this situation, the posture of the second arm is determined according to the posture of the first arm.

This arm-coordination activity can be considered as a coordinator – cooperator type of cooperation in a centralised system, where the first arm acts as the coordinator. From an engineering point of view, this type of coordination activity is always interesting since it may be applied to handle many recently potential robotic cooperative problems in the industrial environment.

A case study is worked out to illustrate this concept. A similar task to that performed by a human is tried to be implemented in a two-arm robot systems. Two robot arms are situated to have a common workspace. A task to constantly hold and move an object in the common workspace is chosen to be trained to the system. This case study is inte

6.3.1 Cooperation of Two Robot Arms

In this case study, the task of two robot arms to hold and move an object is treated fully kinematically without any dynamic consideration. Figure 6.2

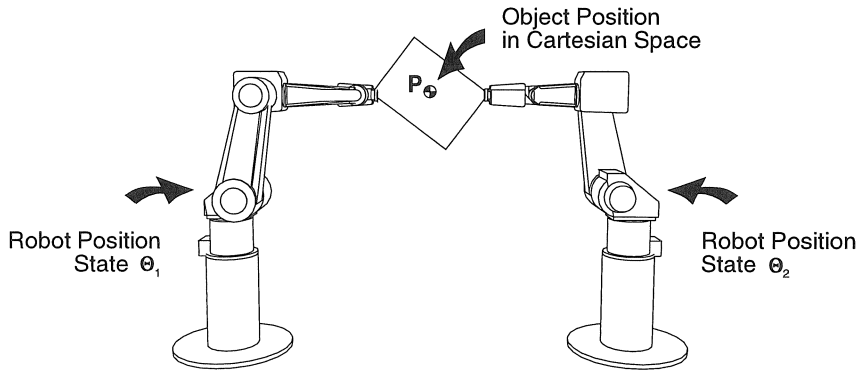


Figure 6.2: Two robot arms holding a common object. $\mathbf{P}$ indicates the object position in the cartesian space and $\Theta = (\theta_1, \theta_2, \theta_3, ..., \theta_n)$ indicates the joint configurations of each n-DOF robot resulting in object position $\mathbf{P}$.

illustrates a two-arm robot system holding an object. The classical kinematic approach to treat the common motion of two robot arms always starts from the planning of the object path in the cartesian space. As the path of the object is planned first, the positions and orientations of the end-effectors are determined so that the motion of the object follows that preplanned path. The

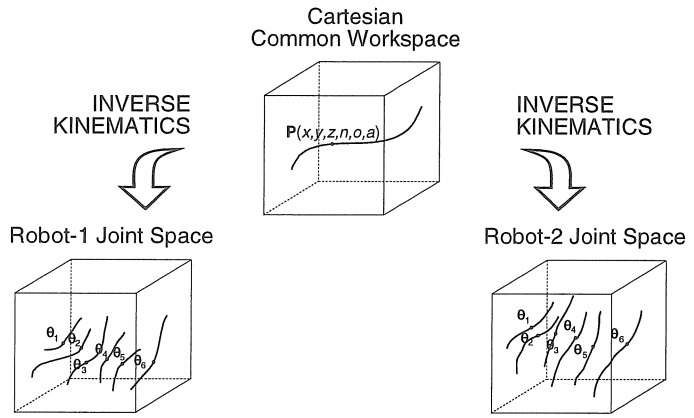


Figure 6.3: Steps to determine the position of both arms if robot kinematic models are fully used. The joint positions of robot-1 and robot-2 are transformed from the object position and orientation in the cartesian space $P(x, y, z, n, o, a)$ by using inverse kinematic model.

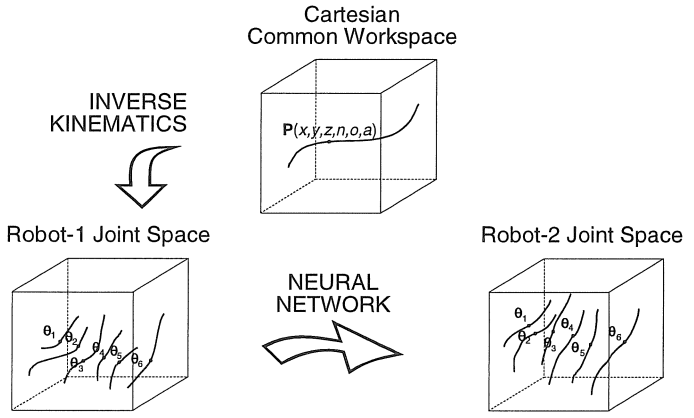


Figure 6.4: A neural network is used to pose the second arm according to the posture of the first arm. This task is done by directly mapping the joint positions of robot 1, obtained by inverse kinematics, to the joint positions of robot 2.

robots' joint configurations therefore must be determined according to the object position at every instant along that path in the cartesian space. The corresponding joint trajectories of the robot arms are obtained by the inverse kinematic solution, as represented in fig. 6.3.

Based on the idea of the coordinator – cooperator type of cooperation, another control scenario is applied. Firstly, the joint trajectories of the first robot arm are computed by the inverse kinematic solution. A direct mapping of the first arm configuration is then established at every instant to obtain the second arm configuration. This mapping, which is clearly non-linear, is learned by a neural network controller, as depicted in fig. 6.4. As a consequence, the transformation between the object trajectories and the second arm joint trajectories is no longer needed. Since the mapping is carried out based on the object positions at every instant during the motion, the joints of each manipulator are controlled in position-control mode.

6.3.2 Robot Environment

The neural network controller is implemented on a simulated robot environment[1]. Two 6-DOF industrial robot manipulators, Kuka 361 and Kuka 160, different in size are used. The simulated robot environment is shown in fig. 6.5. The real dimensions of the simulated robots are presented in the following table, represented by the robots' link lengths L_n.

(mm)	L_1	L_2	L_3	L_4	L_5	L_6
Kuka 361	900	970	1080	0	0	140
Kuka 160	1050	480	645	0	0	120

Table 6.1: Dimensions of the simulated robots.

In this case study, the task of the two-arm robot system is to hold a common object horizontally along the desired trajectory. A common workspace is determined in which the task has to be carried out. A cylindrical object is used. The robots hold the object at both ends (one end-effector at each end), with exactly opposite orientation. The object orientation is kept parallel to an imaginative line connecting both robot bases.

6.3.3 Kinematical Representation of the Cooperative Task

When two robot arms manipulating a rigid object together then a closed kinematic chain is formed. In describing the cooperative task of a two-arm sys-

[1] A robot simulation software package - IGRIP, produced by Deneb Robotics, Inc. is used. The simulation runs on a Silicon Graphics workstation.

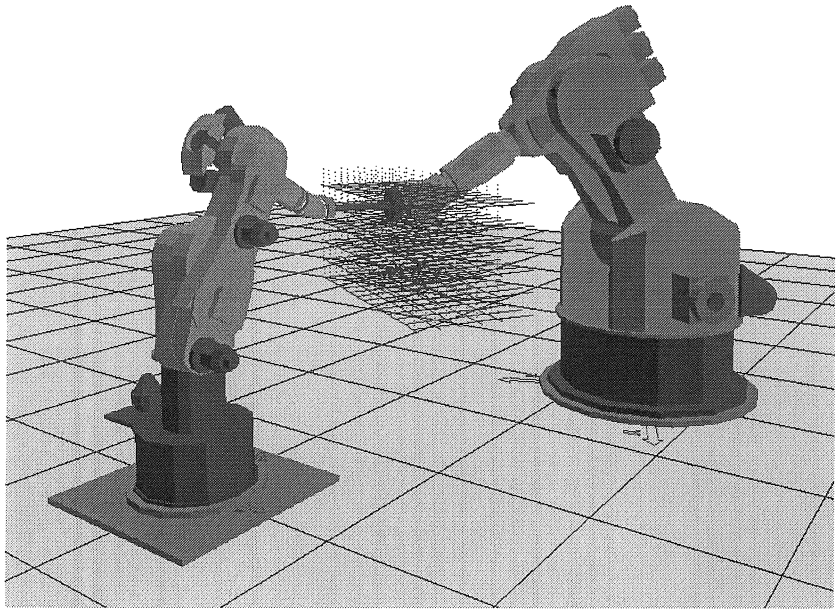


Figure 6.5: The simulated robot environment used in the implementation. Two industrial robot manipulators are used. A common robot workspace to carried out the task is determined. A series of points within that workspace is selected to be used as the object trajectory for collecting the training patterns.

tem, it is necessary to present the position and orientation of each robot end-effector. Also, it is necessary to represent the position and orientation of the object to be manipulated. Homogeneous transformation matrices can be used to describe positions and orientation [Lim and Chyung, 1987]. The structure of the cooperative task then can be defined in terms of homogeneous transformation matrices.

The system configuration to perform the task of moving an object from one place to another with two robot end-effectors, together with its coordinate systems, can be depicted by fig. 6.6. $X(t)$ is the homogeneous transformation matrix of the object coordinate system with respect to the reference coordinate system at time t, as shown in fig. 6.7. $Y^i(t)$ is the homogeneous transformation matrix of the end-effector i ($i = 1$ for the first end-effector, which belongs to Kuka 361, $i = 2$ for the second end-effector, belonging to Kuka 160) with respect to its own base coordinate system at time t. Also, T_r^i is the homogeneous transformation matrix of the base coordinate system of the arm i with respect to the reference coordinate system, and $T_0^i(t)$ is the homogeneous transforma-

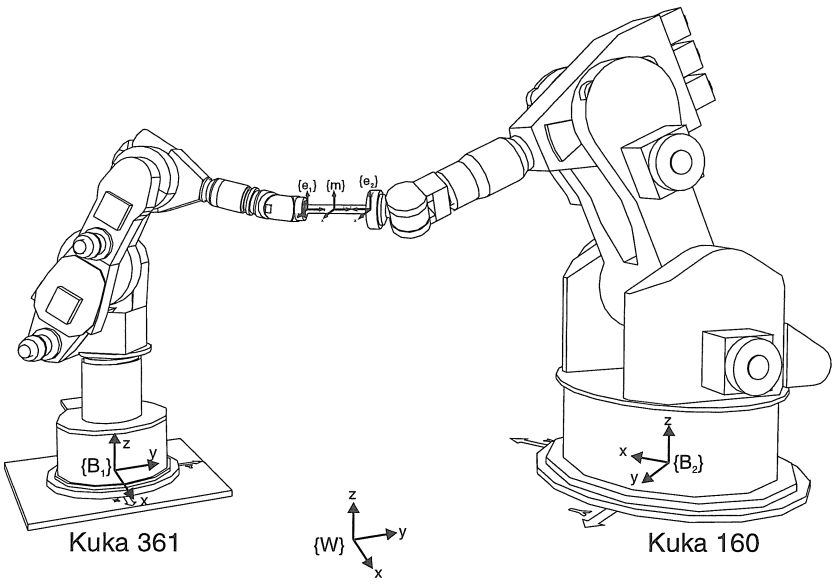


Figure 6.6: The two robot arms used in the simulation holding a common object, and their coordinate frames that are significant in the kinematic transformation.

tion matrix of the end-effector of the arm i with respect to the object coordinate system at time t. The cooperative task is to move the object from its initial position and orientation $X(0)$ to a final place $X(t_f)$, where t_f is the time at which the task is completed. Once the task is given, the values of $X(0)$ and $X(t_f)$ are known. The initial grasping configurations of each robot end-effector $Y^i(0)$, $i = 1, 2$, are also known.

In order to perform a task, the position and orientation of each end-effector $Y^i(t), i = 1, 2$, must be determined for a given $X(t)$. The major difficulty in executing the given task, of course, is the fact that the positions and orientations of the two end-effectors must be such that their relative positions and orientations with respect to each other must be invariant during the execution of the task.

The constraints imposed on the relative position and orientation of the two end-effectors during the execution of the task imply that $T_0^i(t), i = 1, 2$, remain constant for the entire duration. Thus,

$$T_0^i(t) = T_0^i(0) = T_0^i\,, \quad i = 1, 2, \,, \quad \text{for all t } (0 \leq t \leq t_f) \tag{6.1}$$

From the property of the homogeneous transformation matrix, it is easy to see

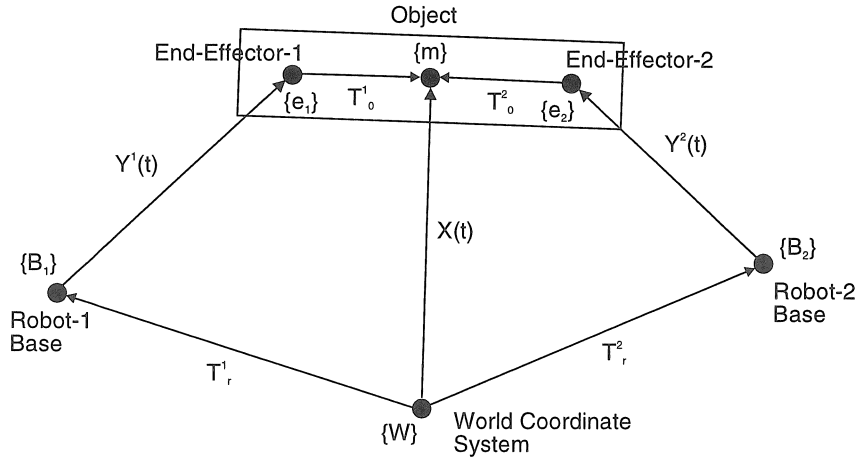


Figure 6.7: Close-chained kinematic scheme of the two robot arms when holding a common object.

that

$$T_r^i Y^i(t) = X(t) T_0^i \,, \quad i = 1, 2, \,, \quad \text{for all t } (0 \leq t \leq t_f) \tag{6.2}$$

For a given $X(t)$, the positions and orientations of the two end-effectors are represented as

$$X(t) = T_r^i Y^i(t) (T_0^i)^{-1} \tag{6.3}$$

Thus, for $i = 1, 2,$

$$T_r^1 Y^1(t) (T_0^1)^{-1} = T_r^2 Y^2(t) (T_0^2)^{-1} \tag{6.4}$$

This relationship can be represented by

$$Y^2(t) = (T_r^2)^{-1} T_r^1 Y^1(t) (T_0^1)^{-1} T_0^2 \tag{6.5}$$

To perform a direct mapping between the two arm positions, the neural network controller then has to be trained to approximate above transformation, so that

$$Y^2(t) = \mathcal{N}\left(Y^1(t)\right) \tag{6.6}$$

where $\mathcal{N}$ is the transformation performed by neural network. For 6-DOF robots,

$$Y^i(t) = f(\Theta_i(t)), \quad i = 1, 2. \tag{6.7}$$

where $\Theta_i(t) = (\theta_1^i(t), \theta_2^i(t), \theta_3^i(t), \theta_4^i(t), \theta_5^i(t), \theta_6^i(t))$ are the joint position values of both arms.

6.3.4 Forward Identification Approach to the Two-Arm Kinematic Mapping

Mapping between two robots' joint configurations is completely non-linear. An error at a joint contributes non-linearly to the final position error at the end-effector. A direct way to build the map is by fully using the arms' kinematic information. The mapping performed by this way is here called the gross-motion mapping. Gross-motion mapping can produce the actual arm posture at every instant. However, involving the information about the interaction between the arms may improve the positioning accuracy. Here, a strategy called fine-motion mapping is applied. The fine-motion mapping is performed by using information about the force generated due to the arm interaction, measured at one of the robot wrists.

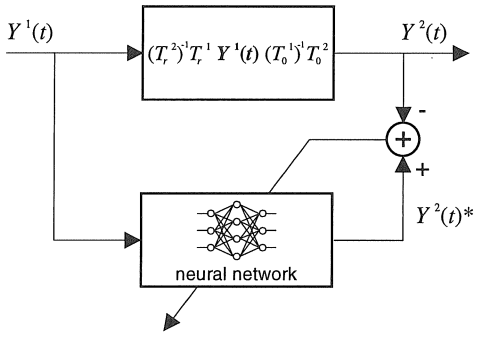


Figure 6.8: Indirect learning approach to approximate the kinematic transformation between the two arms.

The basic learning scheme is based on the forward identification architecture (see section 3.7.1). The learning is done off-line, in which the model of the kinematic transformation is used during the training phase and used as the plant as in fig. 3.14. A neural network is laid-out as shown in fig. 6.8 that allows it to receive the same input $Y^1(t)$ as the model. During training the model outputs the desired response $Y^2(t)$. The training aims at adapting the neural network until its response $Y^2(t)^*$ matches the response $Y^2(t)$ of the model for a given set of inputs $Y^1(t)$. During the training, the norm of the

error vector, $\| Y^2(t)^* - Y^2(t) \|$, is minimised through a number of weight adjustments. In the training sample, $Y^1(t)$ and $Y^2(t)$ are represented by a set of joint position values of the robots at (t), as stated by equation 6.7.

The implementation of forward identification architectures typically applies a multilayer feedforward network with supervised learning algorithms. In this case study, the application of back-propagation and radial basis function (RBF) neural networks are investigated. Those neural network paradigms are described in appendix A.

6.4 Learning Gross Motion Mapping

Gross-motion mapping performs the direct mapping between the kinematical states of the two robot arms at every instant along the trajectory. To produce the joint position values of the second robot arm, this mapping fully relies on the joint positions of the first robot arm.

6.4.1 Training Strategy

To train the neural network controller, a training set consisting of corresponding joint positions of both robot arms is required, computed by using equation 6.5. The joint positions of the first robot are used as the input part of the training patterns, and the joint positions of the second robot are used as the target output part. Those training patterns can be represented as follows :

$$\mathbf{P} = (\Theta_1(t), \Theta_2(t)) \,, \qquad \text{for all t } (0 \leq t \leq t_f) \tag{6.8}$$

where $\mathbf{P}$ is training set of patterns; $\Theta_1(t)$ and $\Theta_2(t)$ are joint positions of the first and the second robot respectively, at one instant which is representing a single object position. Thus the training set patterns are collected from a set of object positions performed between 0 and t_f.

The training set patterns are collected in the simulation. In the simulated robot environment as shown in fig. 6.5, a common robot workspace is determined, which has a "box" shape with a dimension of $1200 \times 400 \times 500$ mm $(length, width, height)$ located within the intersection of two robot workspaces. A cylindrical object with a length of 300 mm is used. A series of points within the workspace is determined to be used as the object trajectory. The points are laid-out in a regular order. Each point has uniform distances to its neighbouring points. By using inverse kinematic transformations of the two robots, the robots are moved to reach those points. At every point, the joint positions of both robots are recorded. After all predefined points have been reached, the collected joint positions are used as the training patterns.

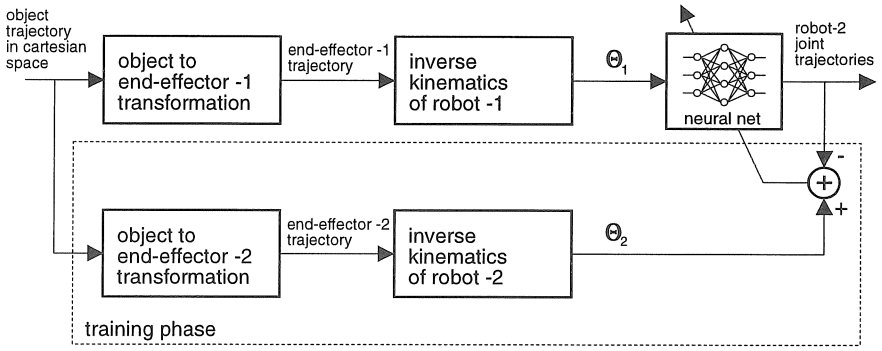


Figure 6.9: Training scheme for the gross-motion mapping.

The training is done off-line as shown in fig. 6.9. There are two separate phases to be carried out. The first phase is the training phase and the second one is the execution phase, which performs the recall of the trained neural network. In this case, the inverse kinematic model of the second robot is needed only during the training phase. After training, the blocks used in the training phase, marked within a dashed-rectangle in the figure, are removed and the trained neural network produces the desired response.

6.4.2 Performance Comparison of Several Network Topologies

Several issues have to be considered in the design of the applied feedforward networks. A particular concern in the use of back-propagation network is overfitting. When the network's output fits the training data too closely, then the overfitting problem arises. In this problem, the network fits the training data very well but it does not generalise to new data (see also section 3.5).

The network risks overfitting the training data because it has too much power. The solution to the overfitting problem is therefore to limit the network's power, to a level which is sufficient to provide a good fit to the training data, but not so much that it risks overfitting. One of the ways to limit the network's power is by limiting the number of hidden neurons. On the other hand, speed is another important attribute in designing a neural network controller. The controller speed is related to the recall time required by a network which is directly depending on the network size. To balance the need of the network's accuracy, speed and the risk of overfitting, the performance of several network topologies with different numbers of hidden neurons are investigated. Their performances are also compared with the performance of an RBF network

topology.

The topological size of the RBF network is determined automatically by the applied orthogonal least squares learning algorithm [Chen et al., 1991]. The number of RBF neurons (hidden neurons) in the network is determined during the learning and is proportional to the number of inputs. Neurons are added to the network – one at an iteration time – until the sum-squared error falls beneath an error goal or until a maximum number of neurons has been reached.

The learning performance is evaluated in terms of learning time and mapping accuracy. The mapping accuracy is evaluated by introducing the used training set and a chosen test set to the trained neural network. The test set corresponds to a series of points within the common workspace which is not a subset of the training set. The accuracy is measured during the trajectory execution, based on the positioning error resulting when holding the object. The error is measured by releasing the object from the end-effector of the first robot while it remains held by the second robot end-effector. The relative position and orientation between the first robot's end-effector and the released object end are measured. The measurement is done based on the first robot's end-effector coordinate frame.

In this comparison, a training set consisting of 330 data and a test set consisting of 73 data are used. Both data sets are collected within the common workspace, and the test set is not a subset of the training set. The evaluation is based on the mean error value and standard deviation of achieved positions and orientations along the trajectories. The sum-squared error of 0.02 is chosen experimentally to terminate the learning of all networks. In the following figure and tables, BP and RBF respectively refer to back-propagation and radial basis function network and notations of $a - b - c - d$ are used to refer to topologies consisting of a neurons on input layer, b neurons on the 1^{st} hidden

Topology		Training Time *(number of epochs)*
BP	6 - 12 - 6	14,052
BP	6 - 60 - 6	6,356
BP	6 - 12 - 6 - 6	not converged after 100,000
BP	6 - 12 - 12 - 6	10,719
BP	6 - 24 - 12 - 6	not converged after 100,000
BP	6 - 24 - 24 - 6	4,033
BP	6 - 60 - 30 - 6	14,481
BP	6 - 120 - 60 - 6	51,861
RBF	6 - 12 - 6	12

Table 6.2: Training time comparison.

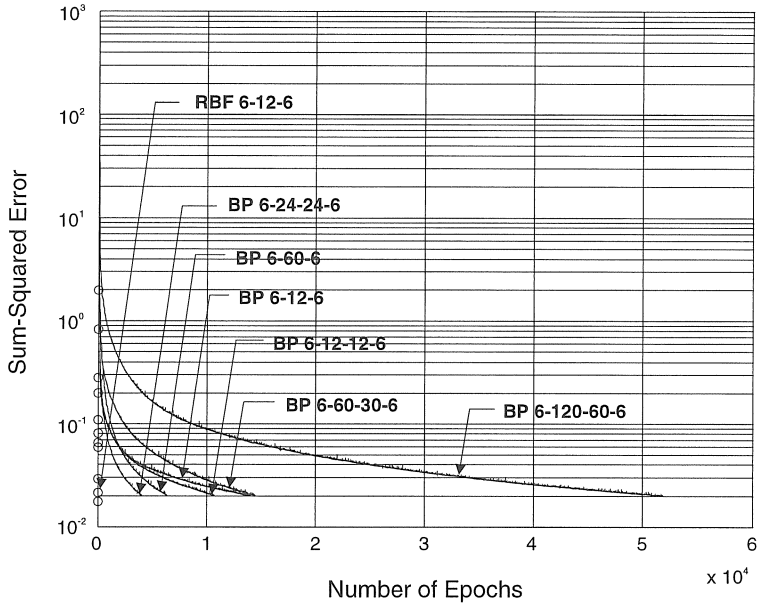


Figure 6.10: Training time comparison of several evaluated architectures. BP and RBF respectively refer to back-propagation and radial basis function networks and notations of $a - b - c - d$ are used to refer to topologies consisting of a neurons on input layer, b neurons on the 1^{st} hidden layer, c neurons on the 2^{nd} hidden layer and d neurons on the output layer.

layer, c neurons on the 2^{nd} hidden layer and d neurons on the output layer[2].

The learning time comparison is presented in fig. 6.10 and table 6.2. Learning time gives no direct effect on the mapping accuracy. The comparison is merely performed to provide an idea about how well the networks can adapt to the mapping complexity. It is shown that the required training time is not proportional with the size of the network. For example, the smallest network, BP 6 - 12 - 6, even needs a training time about as long as the time needed by BP 6 - 60 - 30 - 6, which is much larger in its topological size. In this experiment, the largest network BP 6 - 120 - 60 - 6 obviously takes the longest training time. A significant difference of the required training time is given by the RBF network.

Tables 6.3, 6.4, 6.5 and 6.6 present the mapping accuracy of the trained networks with different topologies. The position and orientation errors are re-

[2]A software package - MATLAB - with the neural network toolbox, produced by The MathWorks, Inc. is used.

Topology		MEAN ERROR - Training Set					
		Position (mm)			Orientation (deg)		
		x	y	z	yaw	$pitch$	$roll$
BP	6 - 12 - 6	5.95	14.00	3.95	-0.49	-0.65	-0.29
BP	6 - 60 - 6	6.32	14.13	4.72	-0.49	-0.67	-0.29
BP	6 - 12 - 12 - 6	6.11	14.02	3.94	-0.48	-0.66	-0.28
BP	6 - 24 - 24 - 6	6.11	14.20	4.14	-0.50	-0.66	-0.27
BP	6 - 60 - 30 - 6	6.00	14.08	4.47	-0.48	-0.65	-0.28
BP	6 - 120 - 60 - 6	6.00	14.02	4.23	-0.48	-0.65	-0.27
RBF	6 - 12 - 6	5.98	14.00	4.01	-0.49	-0.65	-0.29

Table 6.3: Mean error of the training set.

Topology		STANDARD DEVIATION - Training Set					
		Position (mm)			Orientation (deg)		
		x	y	z	yaw	$pitch$	$roll$
BP	6 - 12 - 6	18.51	16.27	13.59	0.63	0.55	0.30
BP	6 - 60 - 6	27.73	38.39	15.78	0.87	0.87	1.41
BP	6 - 12 - 12 - 6	19.75	24.86	12.29	0.90	0.58	0.76
BP	6 - 24 - 24 - 6	26.91	27.77	12.91	0.93	0.91	0.96
BP	6 - 60 - 30 - 6	23.83	29.48	13.61	1.05	0.86	1.13
BP	6 - 120 - 60 - 6	28.36	28.03	12.23	1.05	0.97	1.15
RBF	6 - 12 - 6	13.52	7.25	10.45	0.35	0.28	0.19

Table 6.4: Standard deviation of the training set.

presented by the absolute values of their components, which are the error on x, y, and z axes for position and error of $yaw, pitch$, and $roll$ for orientation. The evaluation is done in terms of these components, because in practice, if error compensation is required, it would be done separately on each of these components.

Tables 6.3 and 6.4 present the mapping accuracy checked by reintroducing the training set as input. It is shown that the mean errors achieved by all networks are almost the same for every measured components, but different in their standard deviations. From this evaluation, the RBF 6 - 12 - 6 network achieves the best accuracy.

A more realistic mapping accuracy is supposed to be presented by the mapping of the test set. This mapping result can also give some indications whether or not overfitting occurs. The mapping accuracy of the test set is presented in tables 6.5 and 6.6.

The occurrence of the overfitting phenomenon can be detected by observ-

Topology		MEAN ERROR - Test Set					
		Position (mm)			Orientation (deg)		
		x	y	z	yaw	$pitch$	$roll$
BP	6 - 12 - 6	7.11	16.31	7.95	-0.55	-0.56	-0.28
BP	6 - 60 - 6	8.27	3.50	12.22	-0.73	-0.34	-0.63
BP	6 - 12 - 12 - 6	3.65	10.44	5.88	-0.95	-0.66	-0.33
BP	6 - 24 - 24 - 6	5.40	13.50	4.46	-0.52	-0.66	-0.43
BP	6 - 60 - 30 - 6	7.99	26.42	-0.32	-0.75	-0.61	-0.04
BP	6 - 120 - 60 - 6	1.20	20.56	2.08	-0.81	-0.73	0.00
RBF	6 - 12 - 6	3.56	11.31	3.86	-0.53	-0.62	-0.27

Table 6.5: Mean error of the test set.

Topology		STANDARD DEVIATION - Test Set					
		Position (mm)			Orientation (deg)		
		x	y	z	yaw	$pitch$	$roll$
BP	6 - 12 - 6	15.75	19.13	14.17	0.26	0.35	0.18
BP	6 - 60 - 6	20.82	35.11	14.16	0.38	0.60	1.50
BP	6 - 12 - 12 - 6	15.02	20.34	10.94	0.89	0.34	0.76
BP	6 - 24 - 24 - 6	28.14	20.07	15.14	0.75	1.04	0.57
BP	6 - 60 - 30 - 6	9.53	22.18	8.11	0.72	0.36	1.09
BP	6 - 120 - 60 - 6	28.90	26.05	7.77	0.80	0.89	1.10
RBF	6 - 12 - 6	9.63	5.25	11.38	0.16	0.22	0.06

Table 6.6: Standard deviation of the test set.

ing both the mapping outputs of training and test sets. If during training the output error of the training set is decreasing while the output error of test set increasing, then overfitting is detected. In can be seen also after the training that if the error obtained from the test set is much higher than the error obtained from the training set, then overfitting occurs.

So far, there is no one network topology that gives a clear evidence about the occurrence of overfitting. Although the test set errors in some output components increase, the overall accuracy of the corresponding network does not always increase. The evaluation is further aimed at the selection of the best network topology in terms of the smallest error produced. Obviously, the RBF 6 - 12 - 6 network gives the best accuracy in both input sets. This network is also the best in terms of training time required, as discussed earlier.

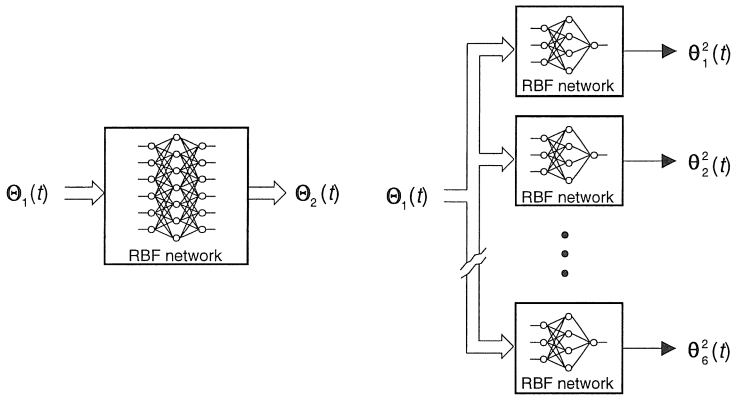
From the processing speed view point, it is clear that the smaller the network size the higher the processing speed that can be achieved. Again, the RBF network meets this speed requirement since it has a compact topology.

In general, the errors produced by the evaluated neural network controllers presented here are quite significant. Therefore, some performance improvement still has to be investigated. At this point, the performance evaluation is concluded by selecting the RBF network to be used in the further investigation.

6.4.3 Verifying the Learning of Joint Dependencies in the Arm Posture Mapping

An arm posture is composed of a set of corresponding joint positions. At a certain arm posture, each joint position is non-linearly dependent upon other joint positions in the same arm, as expressed by equation 6.7. Thereby, the case of posture mapping must be treated as mapping between two whole arm configurations as a unity, as expressed by equation 6.6, and not as a mapping of several separated joint values.

The knowledge of the joint position dependency has to be represented in the neural network controller. The use of one single neural network is the way to let the dependency be learned. A single network can learn to model the joint dependency, since it is provided with the information about all joint states representing a unique arm posture in the form of input-output pairs. Separating the learning of joint values by several independent networks, would cause the joints' unique relationships become invisible for the learning controller and the controller would lose the knowledge about the whole relationships.



(a) a single trained network (b) six separately trained networks

Figure 6.11: Training separation for each output vector component. Instead of mapping $\Theta_1(t)$ to $\Theta_2(t)$ by a single network (a), six networks are trained separately (b) where each network maps $\Theta_1(t)$ to $\theta_n^1(t)$ for $n = 1, 2, .., 6$.

In this section, an experiment is done to verify that procedure. As discussed previously, the neural network controller uses a six-component vector of joint positions $\Theta_1(t)$ as input and produces $\Theta_2(t)$ as output. To verify that the mapping cannot be performed separately for every joint, a network architecture consisting of six independent RBF networks shown in fig. 6.11(b) is applied in the experiment, instead of a single network architecture used previously shown in fig. 6.11(a). The learning is done by decomposing the mapping output vector into six scalar values and by training six separate networks for each output component. The input of each network remains the same, which is a six component vector $\Theta_1(t)$.

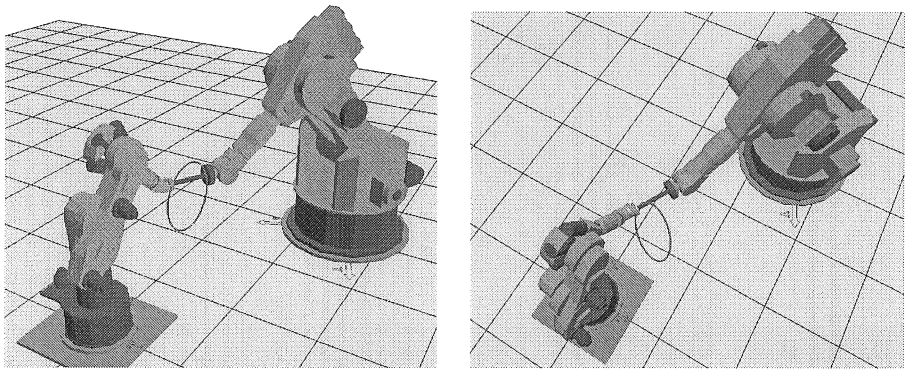


Figure 6.12: The circular-shaped trajectory used as the test set, shown from two different views.

The performance is evaluated by comparing the accuracy of this architecture with the accuracy of the single network architecture in executing a chosen trajectory. Both architectures are trained by the same training set used in the

Architecture	Position (mm)			Orientation (deg)		
	x	y	z	yaw	$pitch$	$roll$
MEAN ERROR						
Single network	3.56	11.31	3.86	-0.53	-0.62	-0.27
Six independent networks	16.28	12.54	4.19	-0.37	-1.00	0.02
STANDARD DEVIATION						
Single network	9.63	5.25	11.38	0.16	0.22	0.06
Six independent networks	17.49	35.56	18.76	0.68	0.46	1.80

Table 6.7: Accuracy comparison between single network and six separately trained network.

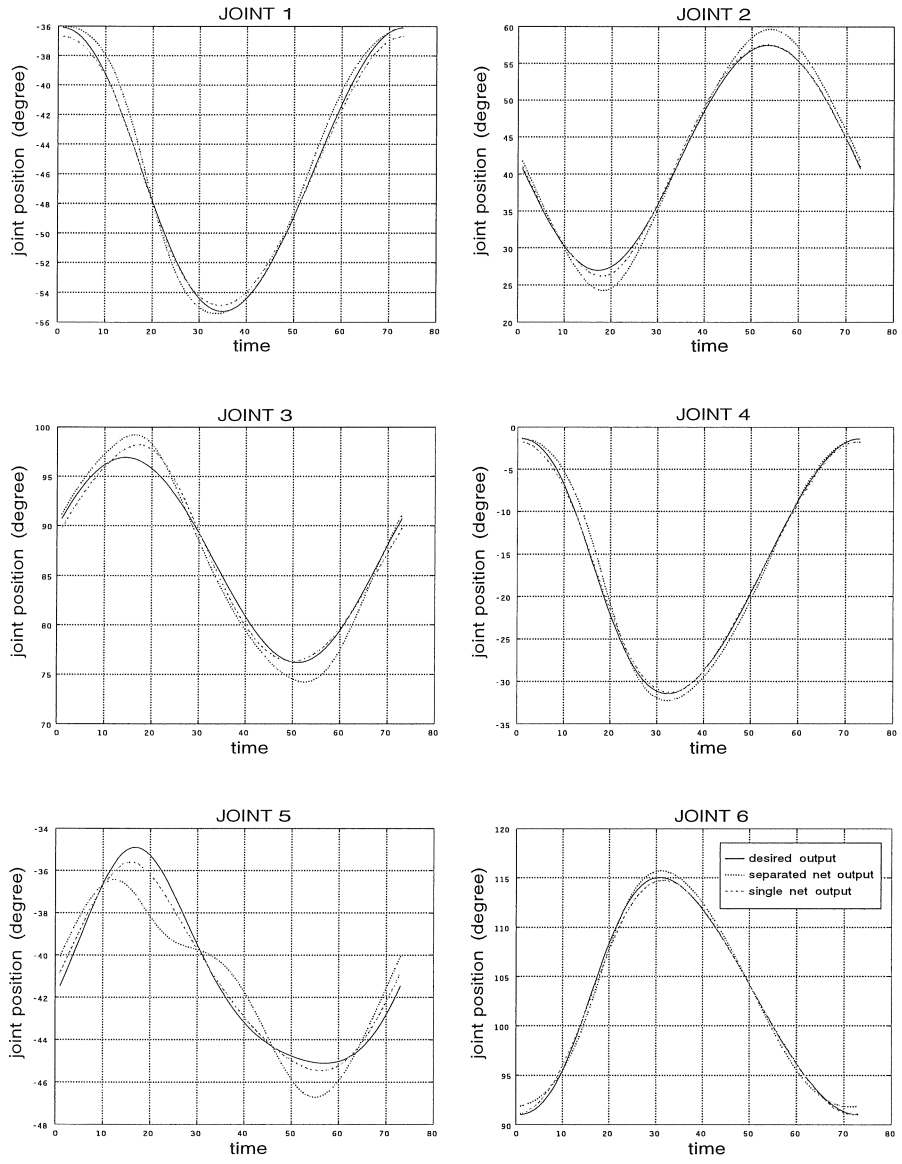


Figure 6.13: Mapping accuracy along the chosen trajectory performed by the single and the separated networks.

previous performance evaluation in section 7.5. The trajectory is constructed by 73 discrete positions which is the same as the one used in section 6.4.2 and has a shape as shown in fig. 6.12. The sum-squared error value of 0.01 is used to terminate the learning phases of all networks.

The result shows that the separately trained network architecture has less accuracy than the single network architecture, as presented in table 6.7. A clear comparison is shown in fig. 6.13 which presents the mapping output of every joint. Generally, the joint positions yielded by the separately trained networks have higher error than the ones produced by the single network architecture. It can be examined particularly on joint 5 whose error is the highest. Joint 5 is a joint on the robot wrist which has a motion range much smaller than the other joints. In this case, the mapping performed by joint 5 is not a one-to-one mapping since one certain position of joint 5 can be related to more than one set of other joints' positions. Only by relying on its input-output pairs, the network would not obtain a good representation about the relationship model it should learn. On the other side, by having all joint states in one set of input-output pairs – which implies that a single network is used – the neural network obtains a complete representation about the joint relationship.

It can be observed that the inaccuracy in the separately trained network architecture arises due to the insufficiency of the information provided by the input-output pairs to learn, since the joint value outputs are treated as independent variables. From the neural network point of view, it should be noted that the relevant consideration of what to learn is not the form of the relationship between the dependent variables in their states, but between the actual inputs and desired outputs – which form a complete arm posture – that should be fed into the learning process.

6.5 Learning Fine Motion Mapping using Force Feedback

In general, the errors produced by the gross-motion mapping are quite significant. In the practical implementation, a stand alone position controller is not sufficient and reliable to control the cooperating arms. Either a passive or active compliant control system is normally supposed to be used. With an active compliant control system, a force control loop is normally applied around the position control loop. In this case, the position control inaccuracy can be compensated by the force feedback [De Schutter, 1986].

A method to incorporate the two-arm interaction force information is investigated. Mapping using force information is assumed to give a better representation about the actual cooperative behaviour performance. In this way,

the mapping can improve the performance of the gross-motion mapping alone. The mapping using force feedback would yield a set of small joint displacements which is added to the output of the gross-motion mapping. Here, this mapping is called fine-motion mapping.

6.5.1 Simulated Force Measurement

In the real robot system, the information about the contact force and torque occurring at the end-effector is obtained from a force/torque sensor mounted on the robot wrist. The force sensor principally works based on the measurement of the deformation occurring in the compliant component in the sensor. If a contact occurs, as a result, the reaction force causes a deformation of the sensor compliance. From this deformation, the corresponding force can be calculated.

A 6-DOF force sensor is normally used to measured force and torque values in all possible directions. The real measured force/torque signals are normally first pre-processed in the following way [Van de Poel et al., 1993] :

- A/D conversion
- subtraction of signal offset which normally occurs in the system and is updated regularly when out of contact
- multiplication with a calibration matrix to yield forces in N and torques in Nmm
- low-pass filtering (user selectable bandwidth)
- transformation from sensor frame to task frame
- gravity compensation.

To incorporate the measured force/torque values (or for short referred to as only forces) in the feedback loop, the force errors (the different between the actual measured values and the desired values) are normally first multiplied by the force feedback gains K_f and then by the compliance matrix $\mathbf{K}_o^{-1}$. This compliance matrix is an estimation of the real contact compliance. The output of these two multiplications is the position error value causing the occurring contact force.

In the case of the simulated force sensing between the two arms, the calculation of the contact force is based on the relative position error occurring in their interaction. The relative position error is translated linearly into the sensor deformation by matrix $\mathbf{K}_{df}$. The contact forces are obtained through multiplication of the calculated deformations with the sensor stiffness matrix $\mathbf{K}_o$. The sensor stiffness matrix represents the model of the compliant force

sensor. This matrix expresses the relation between the force deforming the compliance and the resulting deformation.

Some simplifying assumptions are made in this simulated force sensing. Robot dynamics and disturbances (e.g. joint friction) are disregarded. The robot is considered to be infinitely stiff as compared to the compliance in the force sensor.

6.5.2 Control Scheme and Training Strategy

Basically, the fine-motion mapping aims to determine a small end-effector displacement required to compensate the inaccuracy of the gross mapping output. This small displacement value is assumed depending on, firstly, the occurring contact force between the two end-effectors, and secondly, the current arm configurations that yield that contact force. Based on that assumption, a control architecture depicted in fig. 6.14 is applied.

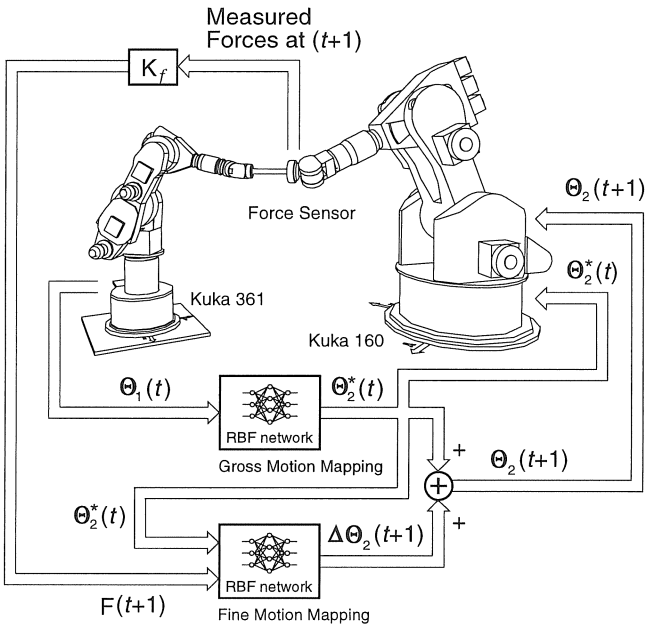


Figure 6.14: Improved control architecture by fine-motion mapping.

Firstly, at instant t, the gross-motion mapper neural network maps the robot-1 joint positions $\Theta_1(t)$ to the gross robot-2 joint positions $\Theta_2^*(t)$, as expressed by the following equation.

$$\Theta_2^*(t) = \mathcal{N}_{Gross}(\Theta_1(t)) \tag{6.9}$$

These joint positions result in a new robot-2 arm configuration that updates the relative position between the two arms. Secondly, a fine-motion loop is executed. The resulting contact forces due to that relative position are measured at the next time instant $t+1$. A force feedback gain K_f is applied to amplify the measured forces. The amplified forces $F(t+1)$ are then introduced to the gross-motion mapper neural network, together with the gross robot-2 joint position values $\Theta_2^*(t)$ that yield the measured forces, as expressed as follows :

$$\Delta\Theta_2(t+1) = \mathcal{N}_{Fine}(\Theta_2^*(t), F(t+1)) \tag{6.10}$$

The fine-motion mapping produces a set of joint displacement values $\Delta\Theta_2(t+1)$ which is added to $\Theta_2^*(t)$ to produced a new improved robot-2 configuration $\Theta_2(t+1)$, as expressed as :

$$\Theta_2(t+1) = \Theta_2^*(t) + \Delta\Theta_2(t+1) \tag{6.11}$$

Two RBF networks are used for both gross and fine-motion mappers.

To train the fine-motion mapper neural network, a training scheme as depicted by fig. 6.15 is applied. The trained gross-motion mapper neural network is used in this training process to provides the actual robot-2 joint configuration which is introduced as input to the fine-motion mapper network.

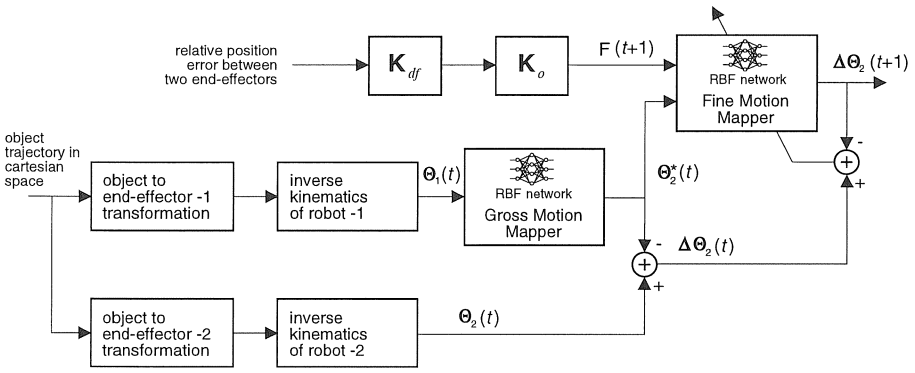


Figure 6.15: Training strategy for the fine-motion mapper neural network.

The complete training patterns are expressed as follows :

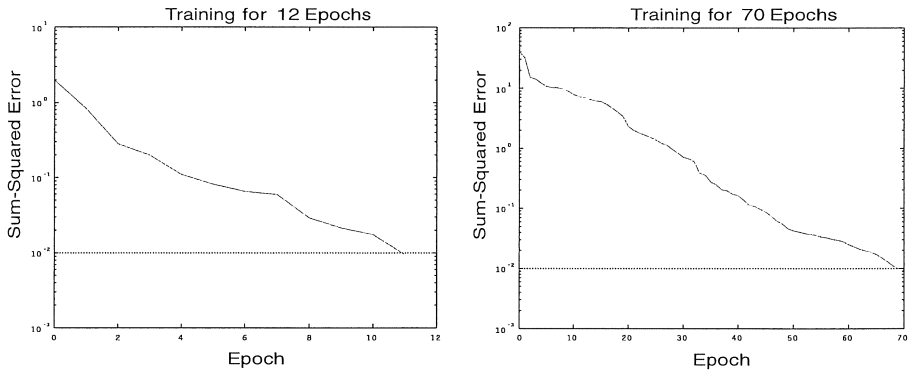
$$\mathbf{P}_i = (F(t+1), \Theta_2^*(t)), \quad \text{for all t } (0 \leq t \leq t_f) \tag{6.12}$$

$$\mathbf{P}_o = (\Delta\Theta_2(t)), \quad \text{for all t } (0 \leq t \leq t_f) \tag{6.13}$$

where $\mathbf{P}_i$ and $\mathbf{P}_o$ are respectively input and output training set patterns; $\mathrm{F}(t+1)$ is the force value vector and $\Theta_2^*(t)$ is the gross robot-2 joint position vector. In the output training set pattern, $\Delta\Theta_2(t)$ represents the compensating displacement values for robot-2 joints. The training set patterns are collected from a set of discrete positions performed between 0 and t_f.

The force values are obtained by multiplying the occurring relative position error between the two arms with the sensor deformation matrix $\mathbb{K}_{df}$ and the sensor stiffness matrix $\mathbb{K}_o$. Since those matrices are linear, for neural networks, their values are not significant. The final input values however will be normalised between -1 and 1. As the output patterns, the difference between gross robot-2 joint configuration and the exact joint position values obtained from kinematic transformations $\Delta\Theta_2(t)$, is introduced.

In the implementation, the same training set patterns as used to train the gross-motion mapping alone (see section 6.4.1) are used. The sum-squared error of 0.01 is applied to terminate the training process. The training progress is shown in fig. 6.16. From the simulations, the optimum size of the RBF networks are found as networks with 12 RBF neurons for the gross-motion mapper and with 70 RBF neurons for the fine-motion mapper. Those imply that the required training time for both networks are also respectively 12 epochs and 70 epochs.



(a) gross motion mapper (b) fine motion mapper

Figure 6.16: Training time consumed by the gross-motion mapper network (a) and by the fine-motion mapper network (b).

6.5.3 Performance Evaluation

A performance evaluation is done to evaluate the improvement yielded by the fine-motion mapping. A spiral-shaped trajectory consisting of 365 discrete positions is used. To check the mapping generalisation ability, some positions lying outside the trained workspace are chosen. The first 300 positions are lying in the workspace and the last 65 positions are lying outside. The view of the trajectory is presented in fig. 6.17.

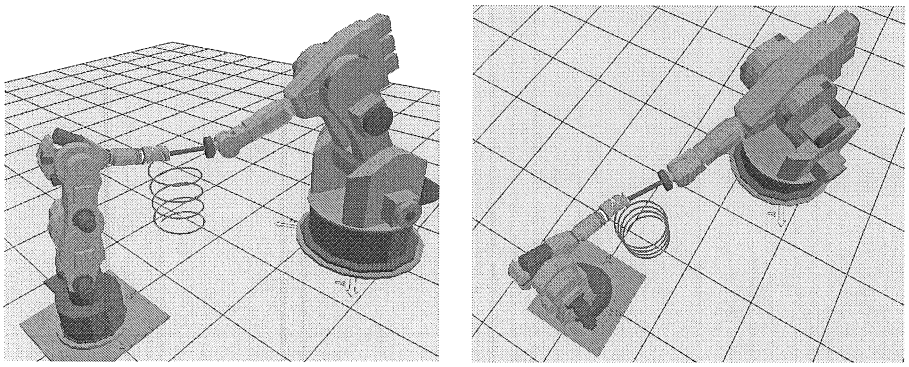


Figure 6.17: The spiral-shaped trajectory used as the test set, shown from two different views.

The accuracy performed by the neural network controller is presented in table 6.8.

Control Scheme	Position (mm)			Orientation (deg)		
	x	y	z	yaw	$pitch$	$roll$
MEAN ERROR						
Gross Motion Mapping	2.21	1.05	6.60	0.10	0.04	0.02
Gross + Fine Motion Mapping	0.19	0.04	-0.05	0.00	-0.03	0.05
STANDARD DEVIATION						
Gross Motion Mapping	14.71	12.31	18.18	0.33	0.36	0.27
Gross + Fine Motion Mapping	2.67	1.61	1.48	0.06	0.18	0.22

Table 6.8: Accuracy of the gross-motion mapping compared with the improved accuracy by the fine-motion mapping.

It is shown clearly by the resulting error that the fine-motion mapping increases the system accuracy considerably. More detailed comparison between

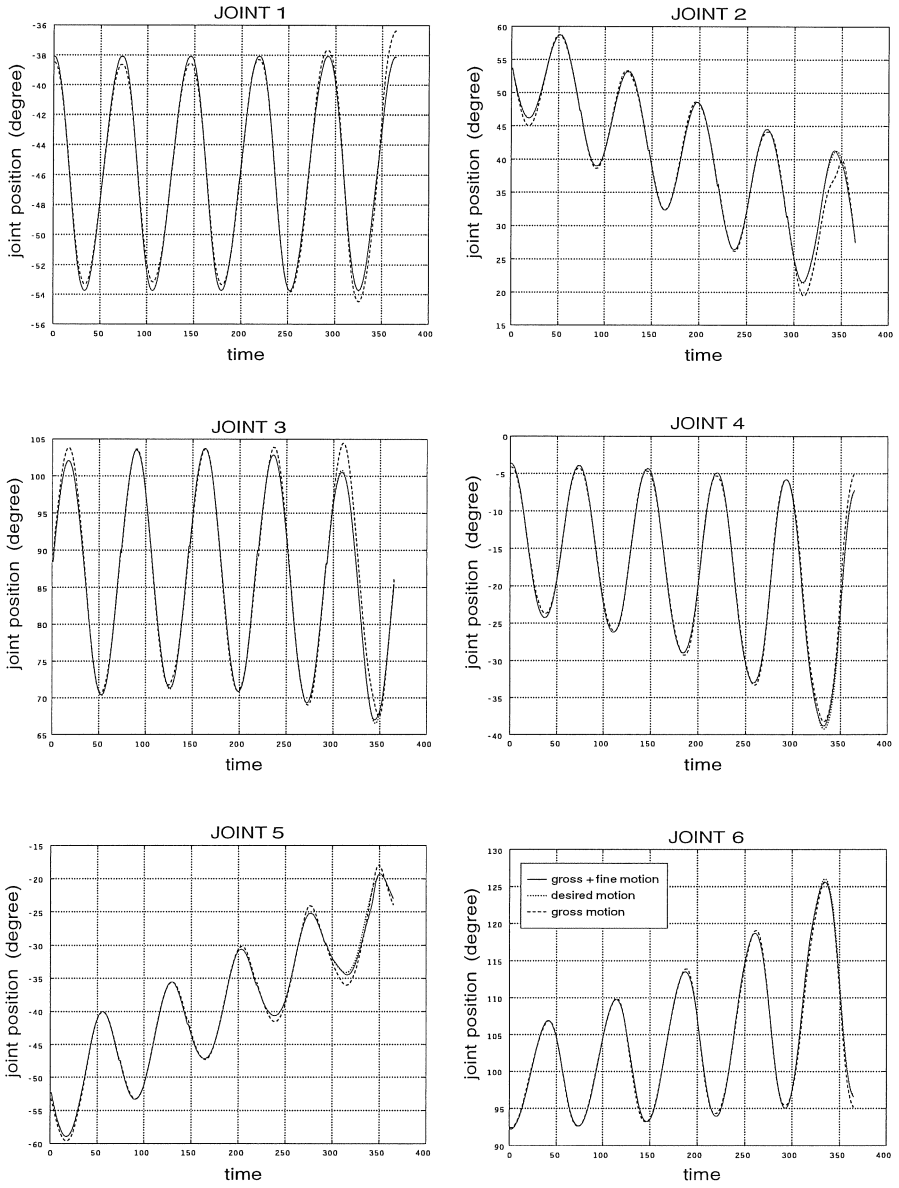


Figure 6.18: Accuracy in joint space along the chosen trajectory performed by the gross motion mapping and the improved performance by the fine-motion mapping.

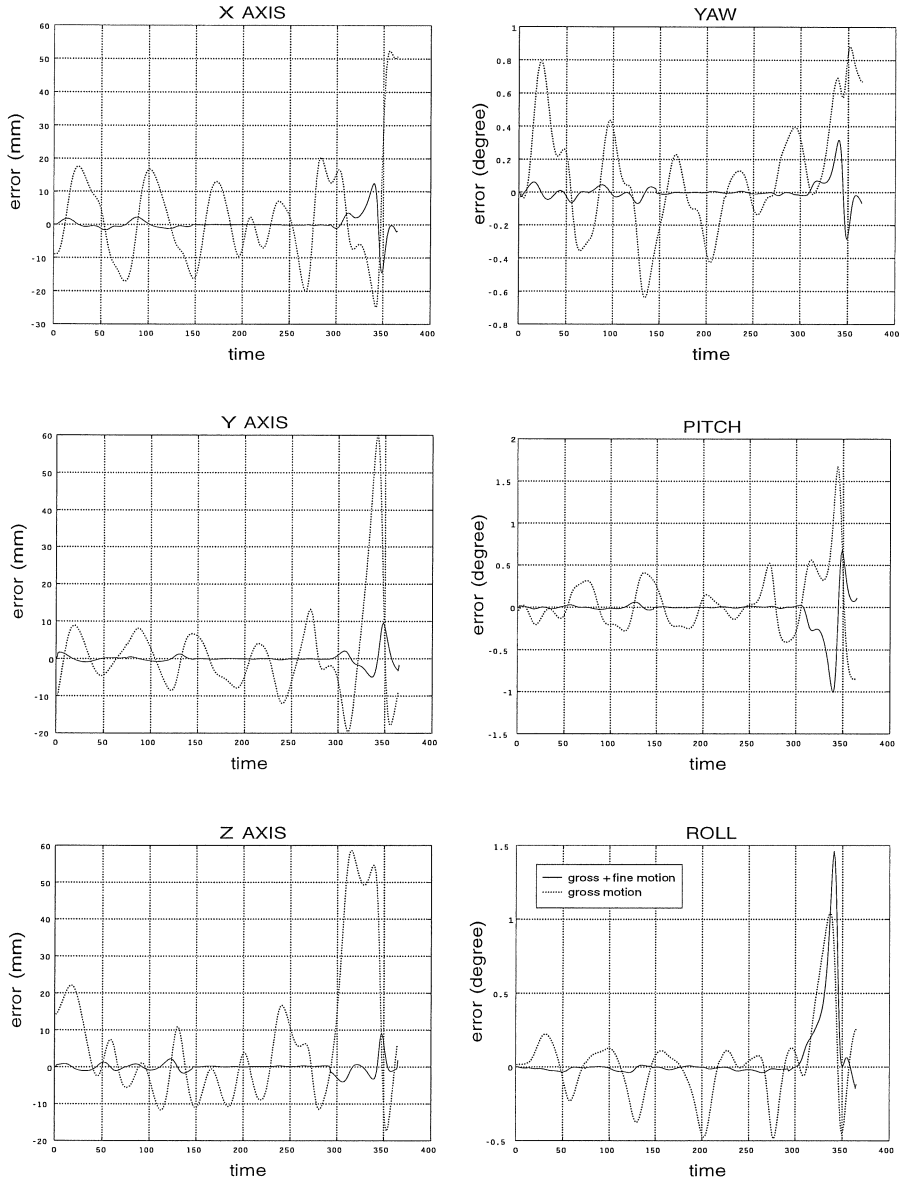


Figure 6.19: Accuracy in cartesian space along the chosen trajectory performed by the gross-motion mapping and the improved performance by the fine-motion mapping.

the accuracy performed by only the gross-motion mapping and the improved accuracy by the fine-motion mapping is presented in fig. 6.18 which shows the accuracy in joint space and fig. 6.19 which shows accuracy of end-effector motion in cartesian space.

In the joint space, it can be seen that the improved joint trajectories by the fine-motion mapping are able to track the desired trajectories very closely and significantly more closely than the trajectories generated by the gross-motion mapping alone. It particularly can be observed in the part of trajectories which are lying outside the trained workspace, that the improved trajectories are still able to track very closely, while the gross trajectories producing quite significant errors.

In the cartesian space, the improvement is shown very clearly. Although the errors produced by both methods during moving outside the trained workspace are larger than the errors inside, on the average the errors of improved mapping method are smaller than the ones produced by the gross-motion mapping alone.

In general, it is obvious that the improved method by fine-motion mapping can perform a better accuracy. From this comparison, the force feedback mechanism shows its effectiveness in achieving a high performance neural network mapping.

6.6 Conclusions

A two-cooperative-arm robot control based on neural network learning has been demonstrated. The principal task of the cooperative behaviour in the case study is to train a robot to be a cooperator (or a follower) of another robot. A fully position-based neural learning is presented, which is here called gross-motion mapping. A position mapping improved by a contact force feedback loop is demonstrated to achieve better accuracy. In that case, a neural network can be trained to find a relationship between the contact forces at a certain arm position to the compensating joint displacements needed to reduce the positioning error based on only gross motion mapping. This compensation mapping is called fine-motion mapping.

The case of arm posture mapping must be treated as mapping between two arm configurations as a whole and not as mapping of several separated joint values. One single neural network can learn the complete joint mapping since it is provided with the information about all joint states representing a unique arm posture in the form of input-output pairs. Separating the learning of joint values by several independent neural networks, causes the joints' unique relationships become invisible for the learning controller and the controller loses the knowledge about the whole relationships.

6.7 Future Developments

Some methods and strategies have been presented in order to perform two-arm coordination. According to the main framework of building a cooperative behaviour through learning as mentioned earlier in this chapter, some further questions could arise at this point. The most logical one could be : "Why do we need to apply learning methods if a full kinematic model can handle the task ?"

First of all, this work is not aimed at finding a new more robust control algorithm, in terms of accuracy or speed, that can replace a kinematic model based approach. The idea we wanted to demonstrate is a new vision for approaching the building of a cooperative behaviour which is leading to letting the robots communicate and build their own rules for interaction, instead of deriving the task for each cooperative agent statically from a same source of information at the top.

This case study indeed does not cover all aspects for building the complex cooperative and coordinative behaviour among robots, but rather demonstrates that multi-robot systems can be controlled by building a model of their relationship and secondly to show that the model of the relationship is not to be always explicitly derived through exact mathematical modelling. There must be something that can be learned from. The relationship can be learned from examples produced by a certain supervision method, for which in this case study, the kinematic models of the robots are used. Certainly, the role of these kinematic models can be replaced by any other supervisory method, as long as this method can represent the expected cooperative behaviour to learn. One example of the supervisory method is by teleoperation, where, in that case, the cooperation rules can be derived and learned directly from real human behaviour.

By putting the emphasis on building the interaction rules in a cooperative environment, the future development of this idea must be more directed towards autonomy, versatility and flexibility. The traditional kinematic approach tends to define the cooperators' individual task statically and rigidly by employing a task planner on the top. On the other hand, the more autonomy, versatility and flexibility are required from an infrastructure, the more the involving agents will have to refer to each other, and less static control approaches must be applied [Van Brussel, 1995]. By referring to the latter, the knowledge about the interaction rules should be posed by every robot involved in a cooperative environment. In that case, training a robot to know its role and task with respect to other robots is a robust way to build intelligent behaviour of a group of robots.

Chapter 7

General Conclusions

Everything should be made as simple as possible,
but not simpler.

ALBERT EINSTEIN

The way how a system coordinates its sensory and motor information reflects the behaviour of that system. There has been considerably discussion in the artificial intelligence and robotics literature in recent years on the acquisition of planning and control behaviours which directly couple input to output and avoid the high computational costs for sensory processing, planning and control. Rather than modularising perception, world modelling, planning and action execution, the new approaches address the building of systems which can integrate perceptual inputs smoothly with motor responses. Robot learning has emerged as a potential concept to build intelligent robots behaviours. This concept may be inspired by the main characteristic of biological sensory-motor coordination which is not rigidly preprogrammed. On the other hand, artificial neural networks have emerged as a promising biologically inspired learning technique. The neural network non-linear mapping capability seems to be potential for handling the non-linear and dynamical issues that characterise mostly the sensory-motor problems.

This dissertation addresses the use of the neural network learning approach to achieve particular perceptive ability and behaviours of robots.

Tactile Holistic Perception by Neural Networks

The available pattern processing techniques limit engineering approaches to machine perception to mostly by analysing the complex sensory signals into components by a set of independent feature-analysing systems.

In fact, as illustrated by human perceptual activities, perception is not a reading by a feature extractor. It is argued that perceptual activities in humans does not proceed by combining evidence about constituent features of a sensory pattern but, rather, uses a holistic process. In order to find better methods for machine perception, some lessons can be learned from this human perceptual characteristic.

Neural networks, by their parallelism, offer a potential ability to allow the holistic process to be applied in machine perception.

From an experimental observation, a direct featureless pattern classification method by a back-propagation and an LVQ neural network can be successfully applied to the tactile contact impressions in a robot gripper. Compared with the more classical feature based methods, this method is more direct and straightforward — demonstrated by its processing time — which is in fact reflecting the holistic nature.

The experimental result also demonstrates that neural network approaches interpret the input as a complete image, and not as a set of statistical data of pixel values. This evaluation can lead to a conclusion that by design, the neural network model already contains the nature of parallel operation, and that can be a good platform to implement the holistic perception.

On the other hand, it can be also observed in the experiment that the applied neural network classifiers are not insensitive to pattern variations in position and orientation. The latter remains a problem for creating a good model for the artificial neural network.

Direction for Future Developments

The presented experimental observation deals with a single sensory information source. A more challenging case for examining the holistic process can apparently be found in applications where more than one sensory information source are available.

Robot Reflexive Behaviour Can Be Built Through Skill Refinement

Reflexive behaviour implies a direct mapping between sensory information and motor command. As illustrated in the case study by a visual tracking control task, a neural network (performed by an action network) can learn to map the non-linear relationship between object states observed by a camera and the joint velocity values required to keep the object image always in the middle of the camera frame.

An action exploration mechanism can be performed in order to search more possible mappings between the sensory space and the action space, based on a reinforcement signal. This mechanism leads to a robot skill refinement ability. Reinforcement-comparison learning algorithms can be used to perform such skill refinement mechanism, which is based on an immediate reward, derived from the visual feedback.

In the case study, it is demonstrated that a gradual improvement of a visual tracking behaviour can be performed based on an immediate reward derived from the visual feedback. To achieve a good skill refinement ability, a number of key aspects and requirements are to be noted :

- **Good exploration strategy,** mainly determined by the accuracy of the dynamic model of the environment built-in in the reward predictor network. This is practically determined by the number of hidden neurons applied in this network.
- **Suitable immediate rewards for the resulting reflexive actions,** determined by the representativeness of the evaluation function.
- **Enough capacity for extending the skill,** determined simultaneously by the number of hidden neurons and the learning rate applied in the action network.
- **System ability to predict object trajectory,** can be built by applying a time-delay network as the action network.

Direction for Future Developments

An interesting topic for the future developments can be in the case where the immediate reward is not available, and instead the environment provides a delayed reward. A delayed reward is generated some time in the future. A system with a delayed reward cannot make the judgement of the current action quality immediately after the generated action affects the world state, but instead the reward will be made at a certain time in the future. In this type of reinforcement learning problem, a good action at one instant does not always mean a good action for the complete mission. Thus, an optimisation has to be carried out during the long-term execution of the action generation.

Two Robots Can Learn to Move Coordinatively Through Good Examples and Representative Feedback

In the case of building a kinematic mapping between two robot arm configurations, a supervised learning method, carried out by a feedforward neural network, has demonstrated the capability to coordinate a robot arm as a co-operator (or a follower) of another robot arm. An RBF neural network shows

a successful training behaviour of the **non-linear joint position relationships** between two 6-DOF articulated robot arms, based on a set of precollected training samples. This fully position mapping is called **gross-motion mapping**.

A position mapping improved by a **contact force feedback** loop is demonstrated to achieve a better accuracy. In that case, a neural network can be trained to find a relationship between the contact forces at a certain actual arm position to the compensating joint displacements needed to reduce the positioning error based on only gross motion mapping. This compensation mapping is called **fine-motion mapping**.

A posture of a robot arm is a unique configuration of its joint positions. This uniqueness can be successfully involved in the learning process if the mapping is learned by a single network. Separating the learning of joint values by several independent neural networks, causes the joints' unique relationships to become invisible for the learning controller.

Direction for Future Developments

Some issues are proposed for future developments. The first is to apply any other supervision method to produce the training samples, for example by human teleoperation which allows a particular cooperative skill to be derived directly from the human operator. Some aspects must be further investigated in this framework such as the network capacity to learn more complex coordinative tasks with more complicated maneuvers.

The second issue is to keep the knowledge about the cooperative behaviour possessed by every robot involved in a cooperative environment. This issue will lead to the investigation of the good representation of the role of a robot agent in its cooperative environment and will contribute to the system autonomy, versatility and flexibility.

Appendix A

The Used Neural Network Paradigms

A.1 Back-Propagation

A good description of the back-propagation learning paradigm is given by [Rumelhart et al., 1986].

The back-propagation paradigm refers to a supervised learning type neural network implemented on a multilayer feedforward network. A feedforward network is a network where each neuron in a layer receives input from every neuron in the previous layer, as shown in figure A.1. As in most models, each neuron computes a weighted sum of all its inputs. Then it applies a nonlinear activation function to the sum, resulting in an output of the neuron. A sigmoid function is the most frequently used activation function in feedforward networks. The sigmoid function is a bounded, monotonic, non-decreasing function that provides a graded, nonlinear response.

In many application, the used sigmoid function is $f(x) = \frac{1}{1+e^{-x}}$. In some of the learning algorithms, the derivative of $f(x)$ is needed to be calculated. The above sigmoid function has a nice derivative $f'(x) = f(x)(1 - f(x))$.

Some networks and training situations have turned out to benefit from a sigmoid function between -1 and 1 instead of the usual 0 and 1 boundaries. The form of $f(x) = \frac{x}{1+|x|}$ has the useful property that it does not involve any transcendental functions. Another alternative is the hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, which approaches its bounds more quickly than the sigmoid function.

The learning by back-propagation technique is accomplished by carrying

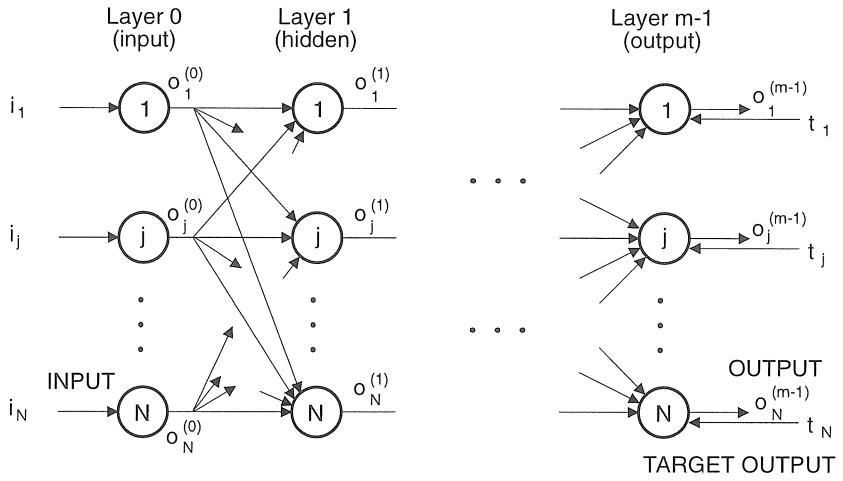


Figure A.1: A multilayer feedforward network.

out two different phases.

In the first phase of the learning algorithm (see figure A.1), the input to the network is provided and values propagate forward through the network to compute the output vector, $\mathbf{O} = (o_1, o_j, \ldots, o_N)$. The output vector of the network is then compared with a target vector, $\mathbf{T} = (t_1, t_j, \ldots, t_N)$, which is provided by a teacher, resulting in an error vector, $\mathbf{E}$.

Basically, the learning aims at finding the minimum error vector $\mathbf{E}$ that can be obtained by searching a set of connection weights. The searching is performed by gradually adapting the connection weights, driven by the value of the error vector $\mathbf{E}$. This is achieved by performing the second learning phase.

In the second phase of the learning cycle, the values of the error vector are propagated back through the network. The error signals for neurons on the hidden layers are determined recursively. For continuous activation functions, the delta learning rule is commonly used (see section 3.6.1). By this rule, error values for layer l are determined from a weighted sum of the errors of the next layer, $l + 1$, again using the connection weights – now "backwards". The weighted sum is multiplied by the derivative of the activation function to give the error value, δ.

Finally, appropriate changes of weights and bias values can be made. The weight change in the connection to unit i in the layer l from unit j in layer $l - 1$ is proportional to the product of the output value, o_i, in the layer l, and the error value, δ_j, in the layer $l - 1$. The bias (or threshold) value, b_i, may be

seen as the weight from a neuron that is always on (produces output value of 1) and can be learnt in the same way.

The algorithm is summarised as follows :

1. Apply the input vector $\mathbf{O}^{(0)} = \mathbf{I}$.
2. Compute the output for each layer :

$$\mathrm{O}_i^{(l)} = f(net_i^{(l)} + b_i^{(l)}) \tag{A.1}$$

where,

$$net_i^{(l)} = \sum_{j=1}^{N^{(l-1)}} w_{ij}^{(l)} o_j^{(l-1)} \tag{A.2}$$

and $N^{(l-1)}$ = number of neurons on layer $(l-1)$.

3. Determine the error vector $\mathbf{E} = \mathbf{T} - \mathbf{O}$.
4. Propagate the error backwards and calculate the elements of the error value vector $\mathbf{D}$.

For neurons on the output layer :

$$\delta_i^{(m-1)} = f'(net_i^{(m-1)} + b_i^{(m-1)})(t_i^{(m-1)} - o_i^{(m-1)}) \tag{A.3}$$

where m = number of layers.

For neurons on the hidden layer :

$$\delta_j^{(l)} = f'(net_j^{(l)} + b_j^{(l)}) \sum_{i=1}^{N^{(l+1)}} \delta_i^{(l+1)} w_{ij}^{(l+1)} \tag{A.4}$$

Here, $f'(x)$ is the derivative of the used activation function.

5. Adjust weight and bias. The changes are calculated by :

$$\Delta w_{ij}^{(l)} = \eta \delta_i^{(l)} o_j^{(l-1)} \tag{A.5}$$

$$\Delta b_i^{(l)} = \eta \delta_i^{(l)} \tag{A.6}$$

And the new values are calculated by :

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) + \Delta w_{ij}^{(l)}(n) + \alpha \Delta w_{ij}^{(l)}(n-1) \quad \text{(A.7)}$$

$$b_{i}^{(l)}(n+1) = b_{i}^{(l)}(n) + \Delta b_{i}^{(l)}(n) + \alpha \Delta b_{i}^{(l)}(n-1) \quad \text{(A.8)}$$

where η, learning rate, and α, momentum factor, are constants that are chosen empirically between 0 and 1. The learning rate determines the "step length" achieved by one learning iteration and the momentum factor is related to the system ability to avoid being trapped in local minima of the error space.

6. Repeat from 1.

The termination of the learning process depends on the allowed minimum error value given by the teacher. An error value which represents the sum-square of the error vector $\mathbf{E}$ is mostly used.

A.2 Radial Basis Function (RBF)

A good description of the radial basis function paradigm is given by [Moody and Darken, 1989].

A RBF neural network is a feedforward network with a single hidden layer, in which the hidden layer consists of RBF neurons and the output layer consists of linear neurons, as depicted in fig. A.2. Adjustable connection weights among the neurons only exist from the neurons in the hidden layer to those in the output layer. The inputs are simply directly connected to the neurons in the hidden layer via unity weights.

The RBF neurons have normalised **Gaussian activation functions**, which are expressed as follows :

$$g_j(\xi) = \frac{exp[-(\xi - \mu_j)^2/2\sigma_j^2]}{\Sigma_k exp[-(\xi - \mu_k)^2/2\sigma_k^2]} \quad \text{(A.9)}$$

where ξ is the input vector itself. By the term *normalised* , it is meant that $\Sigma_j g_j(\xi) = 1$ for any ξ. Thus unit j gives a maximum response to input vectors near μ_j. In this way, it can be said that each hidden unit has its own *receptive field* in the input space, a region centred on μ_j with size proportional to σ_j. The Gaussians are a particular example of **radial basis functions**.

The idea is now to pave the input space (or, the part of it where the input vectors lie) with these receptive fields. Then the problem is almost solved. Suppose a particular input vector ξ^μ lies in the middle of the receptive field

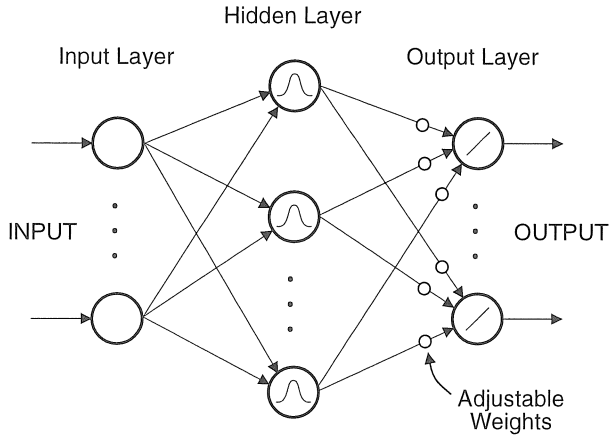


Figure A.2: The RBF network topology

for unit j, so $\xi^\mu = \mu_j$. If the overlaps between different receptive fields are ignored, then only hidden unit j will be activated, making it the only "winner". The output weights leading from that unit can be simply chosen to be $w_{ij} = \zeta_i^\mu$ (for each i), which will produce the target pattern ζ_i^μ at the output assuming linear output units.

If another input lies, say, between two receptive field centres, then those two hidden neurons will be appreciably activated, and the output will be a weighted average of the corresponding targets. In this way, the network makes a sensible smooth fit to the desired function.

The choice $w_{ij} = \zeta_i^\mu$ for the hidden-to-output weights is not optimal if the overlaps between receptive fields are taken into account. Delta rule learning can be simply used to optimise the weights. The rule is expressed as follows :

$$\Delta w_{ij} = \eta(\zeta_i^\mu - O_i^\mu)V_j \tag{A.10}$$

where $O_i^\mu = w_{ij}V_j$. When averaged over μ, this minimises the usual quadratic cost function :

$$H\{w_{ij}\} = \frac{1}{2}\Sigma_{ij\mu}[\zeta_i^\mu - w_{ij}g_j(\xi^\mu)]^2 \tag{A.11}$$

The unsupervised part of the learning is the determination of the receptive field centres μ_j and widths σ_j. Appropriate μ's can be found by any vector quantisation approach, including the usual competitive learning algorithm. The μ's are usually determined by an *ad hoc* choice, such as the mean distance

to the first few nearest neighbour μ's. The performance of the network is not very sensitive to the precise values of the μ's.

A.3 Learning Vector Quantisation (LVQ)

A good description of the learning vector quantisation paradigm is given by [Kohonen, 1987].

A learning vector quantisation (LVQ) neural network is an auto-associative, nearest-neighbour classifier that classifies arbitrary analog spatial patterns $\mathbf{A}_k = (a_1^k, ..., a_n^k), k = 1, 2, ..., m$, into one of p-many classes using an error-correction encoding procedure. It is clearly related to the competitive learning paradigm and an extension of what is referred to as the self-organising feature map – introduced by [Kohonen, 1987]. The LVQ strengths include its ability to perform non-parametric pattern classification and provide real-time nearest-neighbour response. Another capability is that it can allocate reference vectors to the centroids of the decision regions without any *a priori* information concerning the distribution of data.

Competitive learning networks in general learn to recognize classes of similar input vectors in such a way that neurons physically close together in the neuron layer respond to similar input vectors. However, the classes that the competitive layer finds, are dependent only on the distance between input vectors. If two vectors are very similar, the competitive layer will probably put them into the same class. There is no mechanism in a competitive layer design to dictate whether two arbitrary input vectors are in the same class or in different classes. LVQ networks, on the other hand, learn to classify input vectors into target classes chosen by the user. The target classes are introduced together with the corresponding input patterns during a training process.

The LVQ network learns off-line, operates in discrete time, and is represented by a two-layer feedforward topology as shown in fig. A.3, where the n neurons in the layer 1 correspond to $\mathbf{A}_k$'s components, and the p neurons in layer 2 each represent a pattern class.

This two-layer neural network classifies patterns by finding the optimal set of reference vectors – where the reference vector is the set of connections that abut on neuron in layer 2 – for a given training set. The reference vectors are stored in the weight matrix $\mathbf{W} = (w_{11}, ..., w_{np})$. During operation, the neurons in layer 2 employ an invisible on-centre/off-surround competition that is used to choose the proper class for the presented input. These lateral interactions are shown in the figure as shaded self-exciting/neighbour-inhibiting connections to emphasize this point. For the sake of figure clearness, not all connections are shown. There is actually a connection from each neuron in

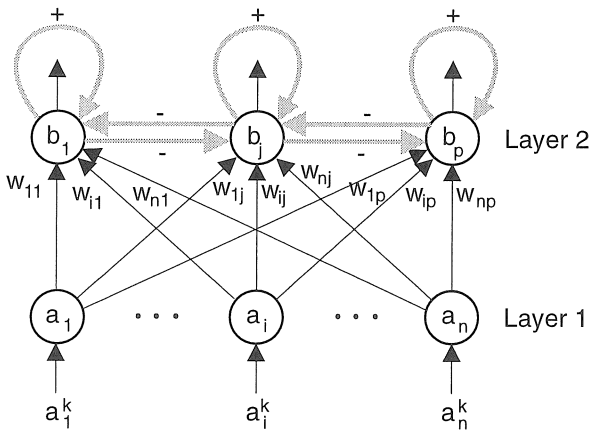


Figure A.3: The LVQ network topology

layer 1 to each neuron in layer 2, a shaded lateral connection from each neuron in layer 2 to every other neuron in the same layer, and a shaded positive recurrent connection from every neuron in layer 2 to itself.

Learning Algorithm

For the single winner unsupervised learning – which allows only one output neuron (in layer 2) to be active during the learning phase – the learning procedure automatically determines the p-best reference vectors needed to represent the space spanned by a given set of data vectors. Since only one output neuron is active during the learning phase, hence only the connections that abut the active layer 2 neuron are adjusted. The learning procedure is outlined as follows :

1. Initialise all the layers 1 and 2 connection weights to some random value in the range [0,1].

2. The connections that emanate from layer 1 and abut the j^{th} layer 2 neurons form the weight vector $\mathbf{W}_j$. For each pattern $\mathbf{A}_k$, $k = 1, 2, ..., m$, the following steps are performed :

   - Find $\mathbf{W}_j$ closest to $\mathbf{A}_k$

$$\| \mathbf{A}_k - \mathbf{W}_g \| = \min_{j=1}^{p} \| \mathbf{A}_k - \mathbf{W}_j \| \tag{A.12}$$

where $\mathbf{W}_g$ is the $\mathbf{W}_j$ closest to $\mathbf{A}_k$ and the Euclidian distance between any two real-valued n-dimensional vectors $\mathbf{X}$ and $\mathbf{Y}$ is defined as

$$\| \mathbf{X} - \mathbf{Y} \| = \left[\sum_{i=1}^{n} (x_i - y_i)^2 \right]^{1/2} \tag{A.13}$$

This operation is essentially a competition amongst the layer 2 neurons with the largest activation (closest reference vector to the data vector) remaining the winner (see section 3.6.2 for a description of competitive learning).

- Move $\mathbf{W}_g$ closer to $\mathbf{A}_k$ using the equation

$$\Delta w_{ij} = \alpha(t) \left[a_i^k - w_{ig} \right] \tag{A.14}$$

for all $i = 1, 2, ..., n$, where Δw_{ig} is the connection strength from the i^{th} layer 1 neuron to the g^{th} layer 2 neuron, and $\alpha(t)$ is the learning rate at time t defined as

$$\alpha(t) = t^{-1} \tag{A.15}$$

3. Repeat step (2) for $t = 1, 2, ..., z$.

Recall Algorithm

LVQ recall determines the class, b_g – represented by the weight vector $\mathbf{W}_g$ – that the input pattern A is most closely associated with. $\mathbf{W}_g$ is determined by finding the closest $\mathbf{W}_j$ – in Euclidian distance – to A. In essence, the layer 2 neurons all compete with each other and the largest layer 2 neuron activation prevails while all others are quenched. Hence, at the end of the competition, the layer 2 neuron representing the proper class will have a value of 1 and the other layer 2 neurons will have the value 0. This action is illustrated by the equation

$$b_g = \begin{cases} 1 & \text{if } \| \mathbf{A} - \mathbf{W}_g \| = \min \| \mathbf{A} - \mathbf{W}_j \| \\ 0 & \text{otherwise} \end{cases} \tag{A.16}$$

where b_j is the j^{th} layer 2 neuron's value and A is the input vector.

Appendix B

The Used Statistical Pattern Classification Rules

The statistical pattern classification rules described hereafter are used in chapter 4.

B.1 Nearest-Neighbour Rule

The nearest-neighbour rule assigns an unknown observed $\vec{x}$ to a class S_j according to the following expresion :

$$\vec{x} \in S_j \quad \text{if} \quad d(\vec{x}, S_j) = \min_k \; d(\vec{x}, S_k) \tag{B.1}$$

where,

$$d(\vec{x}, S_k) = \min_{m=1..M_k} \; d(\vec{x}, \vec{y}_m^{(k)}) \tag{B.2}$$

By simplification, the Euclidean distance $d(\vec{x}, \vec{y}_m^{(k)})$ can be calculated by the following expression :

$$d(\vec{x}, \vec{y}_m^{(k)}) = \sqrt{\sum_{n=1}^{N} \{x_n - Y_{mn}^k\}^2} \tag{B.3}$$

B.2 Bayes Rule

The Bayes rule classifier is expressed by the following equation :

$$\vec{x} \in S_j \quad \text{if} \quad g_j(\vec{x}) = \max_{k=1..4} g_k(\vec{x}) \tag{B.4}$$

The Gaussian distribution function $g_k(\vec{x})$ is expressed by the following equation :

$$g_k(\vec{x}) = -\left\{\log(|\Phi_k|) + (\vec{x} - \vec{\mu}_k)^t \cdot \Phi_k^{-1} \cdot (\vec{x} - \vec{\mu}_k)\right\} \tag{B.5}$$

where the mean values $\vec{\mu}_k$ and the covariance matrices Φ_k for class k are calculated as follows :

$$\vec{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{ki} \tag{B.6}$$

$$\Phi_k = \frac{1}{N_k} \sum_{i=1}^{N_k} x_{ki} x_{ki}^{'} - \vec{\mu}_k \vec{\mu}_k^{'} \tag{B.7}$$

where N_k denotes the number of patterns in class k, and x_{ki} represents the i^{th} pattern in the class k.

From the used sample data, the mean values for each class $\vec{\mu}_k$ are obtained as follows :

$$\vec{\mu}_1 = \begin{bmatrix} 0.73 \\ 0.71 \end{bmatrix} \tag{B.8}$$

$$\vec{\mu}_2 = \begin{bmatrix} 0.25 \\ 0.08 \end{bmatrix} \tag{B.9}$$

$$\vec{\mu}_3 = \begin{bmatrix} 0.37 \\ 0.23 \end{bmatrix} \tag{B.10}$$

$$\vec{\mu}_4 = \begin{bmatrix} 0.13 \\ 0.07 \end{bmatrix} \tag{B.11}$$

and the covariance matrices for each class Φ_k are obtained as follows :

$$\Phi_1 = \begin{bmatrix} 16 & 16 \\ 16 & 28 \end{bmatrix} \cdot 10^{-3} \tag{B.12}$$

$$\Phi_2 = \begin{bmatrix} 2 & -0.7 \\ -0.7 & 4.7 \end{bmatrix} \cdot 10^{-3} \tag{B.13}$$

$$\Phi_3 = \begin{bmatrix} 1.3 & 0.2 \\ 0.2 & 3.2 \end{bmatrix} \cdot 10^{-3} \tag{B.14}$$

$$\Phi_4 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 2.1 \end{bmatrix} \cdot 10^{-3} \tag{B.15}$$

Appendix C

Simulation Environment for Learning Visual Tracking

This simulation environment is dedicated to the development and optimisation of the robot visual tracking learning control, which is specially developed to facilitate the work discussed in chapter 5. This simulator program is self-written by using C-languange and to be executed in the X Window/HP-UX[1] UNIX environment. This appendix describes the structure of the simulator program, as well as its design considerations.

C.1 Some Design Considerations

One of the important features of an environment to develop a learning based control algorithm is that it provides a facility for the developer to monitor the evolution and the adaptivity of the learning agent and to access easily the learning parameters that contribute to its learning performance.

Since learning is characterised by changes, it is very important to have a facility to observe whether or not those changes are going towards the desired behaviour. It is a generally accepted fact that a visually presented changing state is more easily understood and followed than a textual one. Also, visual material instills more confidence in the observer.

Instead of monitoring a set of numerical values or charts that reveals the progress of the behaviour development and the learning state, it is more convenient to visually observe how the robot arm can track the object. Thus, an

[1] X Window is a network-based windowing system, firstly released by the Massachusetts Institute of Technology in 1987. In this application, the X Window system runs on a Hewlett-Packard's UNIX operating system : HP-UX.

intuitive judgement can directly be made, and changes of behaviour-affecting learning parameters can be made interactively.

One supporting feature in a neural network simulation, if it is further followed by an implementation on the real system, is that of the interchangeability of the optimised weight matrix structure and learning parameters between the simulation environment and the real robot system. In this development the simulation program can output values that can be directly used, through files, by the real robot controller.

Besides its main use as an optimisation tool for learning behaviour, this simulation program has also been used as an educational tool for students to have a quicker understanding about the working principle of machine learning. Finally it provides hands-on experience about the effect of particular parameters or configurations to the overall learning performance.

C.2 Simulator Structure

Based on the need for an adaptive learning control simulator, some basic requirements are considered and defined. The requirements invoke the simulator program to provide the following items :

- A facility to monitor the evolution and the adaptivity of the learning agent.

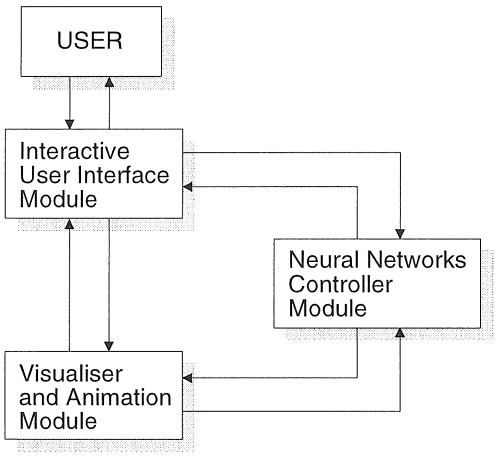


Figure C.1: Modules that construct the overall simulation environment and their interaction.

- A facility to access easily the learning parameters that contribute to the learning performance.
- A facility to interchange the optimised weight matrix and learning parameters between the simulation environment and the real robot system.

Those requirements encourage to build a form of interactive simulator. An X Window environment under a UNIX operating system is chosen. By utilising the existing libraries provided by *X library* or Xlib, the required interactive simulator can be developed.

The whole simulation task is established by a collaboration of three separate modules, shown in fig. C.1. All modules share data and communicate to each other. Interactions with the user are handled by the interactive user interface module. This module provides interactive menus activated by mouse clicks and reads user's input files. It also reads mouse position and mouse movement on screen; these are used by the user to indicate locations or certain object movement trajectories.

The visualiser and animation module handle the real-time generation of simulation output images. It displays the robot behaviour in terms of robot movement with respect to the object, robot states, the evaluation of the current world situation, and tracking error.

The neural network control module runs the robot control algorithm.

C.2.1 X Window Environment

X Window system is a network based windowing system, that allows applications to run in a network of different systems. While many applications can execute locally on a workstation, other applications can execute on other machines, sending requests across the network to a particular display and receiving keyboard and pointer *events* from the system controlling the display. Events include user input (keypress, mouse click, or mouse movement) as well as interaction with other programs.

The program that control each display is known as a *server*. The server acts as an intermediary between user programs (called *clients* or *applications*) running on either the local or remote systems and the resources of the local system. The server performs the following task [Nye, 1992] :

- Allows access to the display by multiple clients.
- Interprets network messages from clients.
- Passes user input to the clients by sending networks messages.
- Does two-dimensional drawing. Graphics are performed by the display server rather than by the client.

- Maintains complex data structures, including windows, cursor, font, and "graphics contexts" as *resources* that can be shared between clients.

Another important concept in X programming is that applications do not actually control such things as where a window appears or what size it is. In this case, clients must not be dependent on a particular window configuration. Instead, a client gives *hints* about how long and where a window would like to be displayed. The screen layout or appearance and the style of user interaction with the system are left to a separate program, called the *window manager*.

Applications communicate with the server by means of calls to a low-level library of C-language routines known as *Xlib*. Xlib provides functions for connecting to a particular display server, creating windows, drawing graphics, responding to events, and so on. Xlib calls are translated to protocol requests sent via a network protocol either to the local server or to another server across the network.

The window manager is actually another program written with *X library* or Xlib, that is given special authority to control the layout of windows on the screen. The window manager typically allows the user to move or resize windows, start new applications, and control the stacking of windows on the screen, but only according to the window manager's window layout policy. A *window layout policy* is a set of rules that specify allowable sizes and positions of windows and icons.

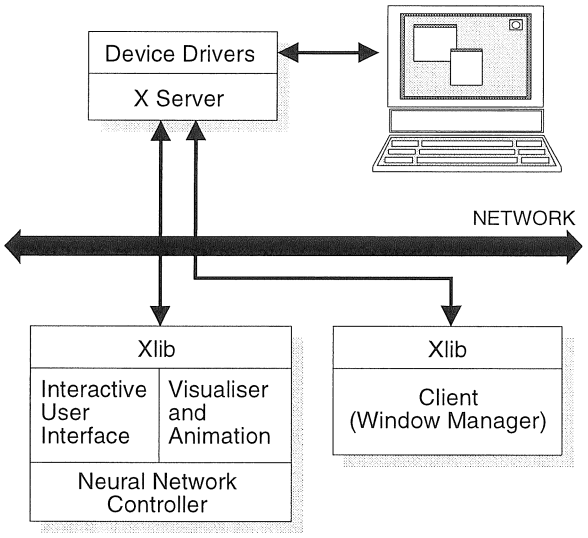


Figure C.2: The simulator program within the X Window system architecture.

As a client within the X Window system architecture, the simulator can be depicted as in fig. C.2. The simulator program, as a client, sends the X Window system protocol requests by means of Xlib functions. The requests are received by the X Server, which is a program that runs on each system that supports a graphics display, keyboard, and mouse. X Server then passes back events and replies to clients via the network.

C.2.2 Control Algorithm Implementation

To execute the whole control scheme, a data flow is established as shown in fig. C.3.

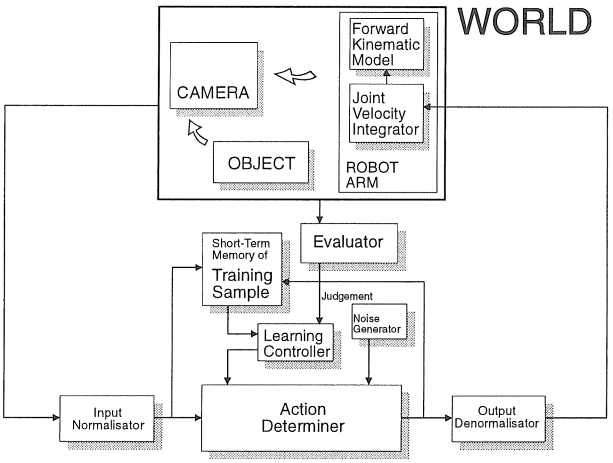


Figure C.3: Data flow during the action phase.

The action network and the noise adder are together considered as an action determiner. Normalisation and denormalisation procedures are done to handle the conversion between the real world values and the network internal values. When the action determiner decides an action, the input and output data are stored in a short-term memory. These will be used further to train the action control network. By using the ouput of the prediction network as argument, noise is generated and added.

In the evaluation phase, the evaluation is done after every single robot movement. A robot forward kinematic model is needed to compute the movement that produces the new state. A subroutine implementing the forward kinematic model is in charge of producing the new states by using the position values of joints 1 and 2 as its arguments. These joint positions are calculated by a joint velocity integrator, which integrates the joint velocities — the output

of the action determiner — to provide the current robot position.

C.2.3 User Interface

The user interface handles program interaction with the user. User commands are given directly through an interactive menu by mouse clicks. The mouse is also used to introduce the motion trajectory of the moving object directly on the display. The neural network controller specification and the stored weight matrix are given through 2 different files. On the other hand, the simulation output is shown through several visualiser windows.

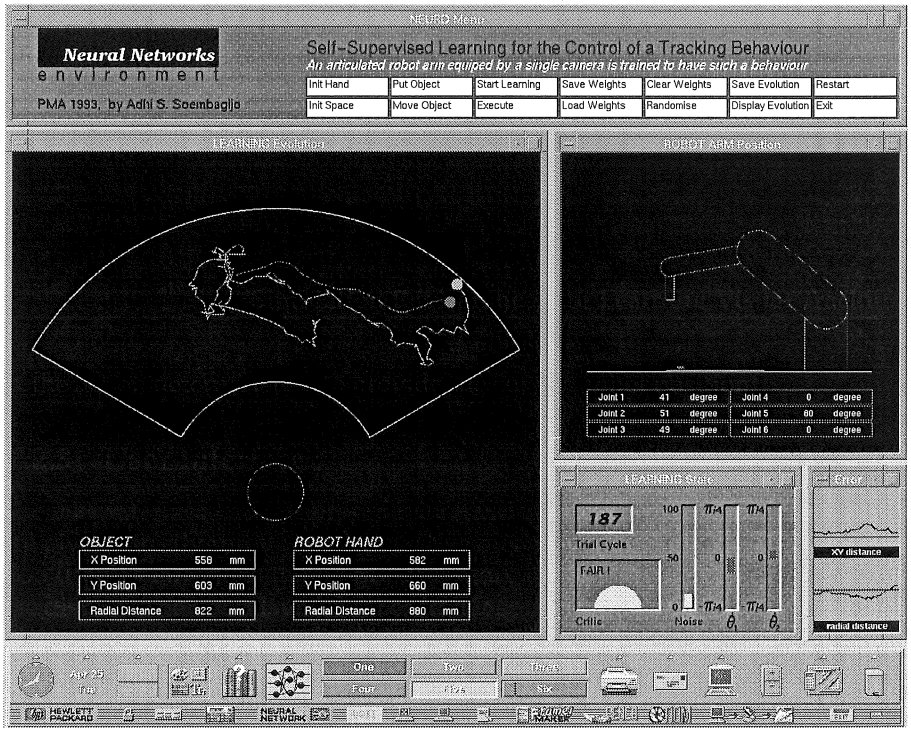


Figure C.4: The simulator user interface.

Figure C.4 shows the user interface of the simulator program. The window labelled NEURO Menu collects the user command through buttons. The main window displayer, labelled LEARNING Evolution displays the top view of the robot hand motion within its workspace area. This window also shows the values of object and robot hand position with respect to the robot base. The initialisation of the object and robot hand positions are done by locating the

mouse cursor to the desired positions followed by left button clicks. During the execution, the object then can be moved by dragging the mouse while the left button pressed. Another window, labelled ROBOT ARM Position shows the robot arm configuration, both by image and numerical values of joint positions. The window labelled LEARNING State shows some significant internal states in the learning algorithm. They are the evaluation result of the current action (critic or judgement), the added noise, and the joint velocity output. Also the trial cycle, which refers to the number of action-prediction iteration, is shown. The ERROR window shows the position error of the robot hand with respect to the tracked object.

The use of the interactive user interface can be described by fig. C.5. The provided functions and operations are used in three different phases : initialisation, execution, and termination of the learning iteration.

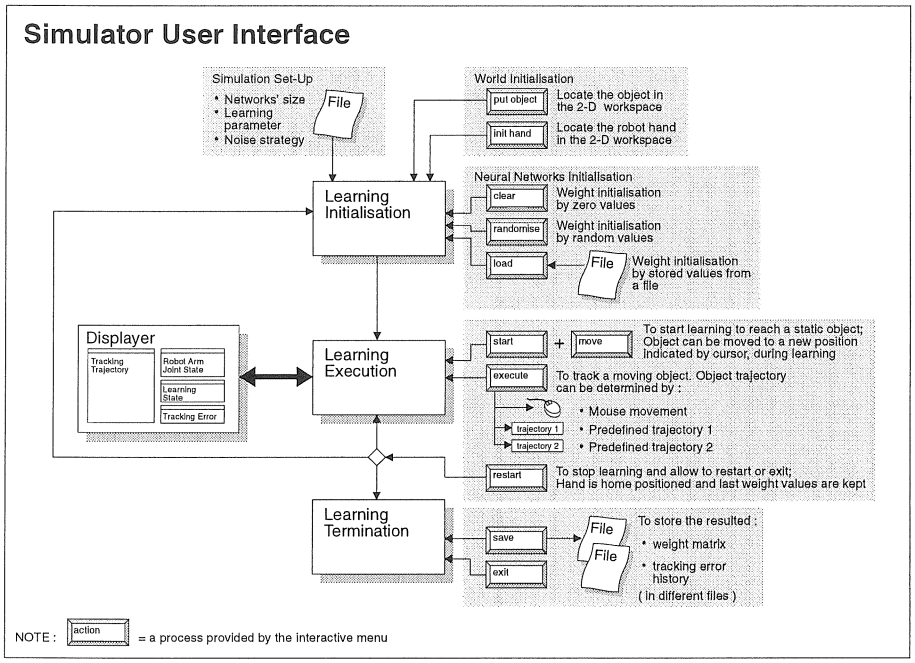


Figure C.5: The functional scheme of the simulator user interface.

Learning Initialisation

The initialisation is done in two parts : world initialisation and neural networks initialisation. Besides them, a description about the controller configu-

ration has also to be given. The description is given through a file containing the neural networks' size, learning rates and the noise control strategy. An example of the file content looks as follows :

```
lr_act          = 0.9
hidl_act        = 2
hidn_act        = 18 6

lr_pred         = 0.5
hidl_pred       = 1
hidn_pred       = 20

noise_strategy = 1
```

`lr_act` and `lr_pred` refer to the learning rates for both action and prediction networks. `hidl_act` and `hidl_pred` specify the number of hidden layers, and `hidn_act` and `hidn_pred` specify the number of neurons in the hidden layers. `hidn_act = 18 6` means that the first layer consists of 18 neurons and the second layer consists of 6 neurons. `noise_strategy` specifies the chosen noise control strategy (1 or 2).

The neural networks initialisation is in fact the initialisation of the connection weights, which can be done either through menu buttons or through a file. The zero or random initialisation are done by clicking the related menu button and initialisation by prestored values is done through a file. The world initialisation is the determination of the object and robot hand position in the workspace. Those are done by first pressing the related button, and then locating the mouse cursor followed by a mouse left button click.

Learning Execution

The learning execution is started by clicking the Start Learning button. During learning, the static object can be moved to another position by clicking the Move Object button followed by a cursor indication at a new desired object position. Learning to track the moving object is started by clicking the Execute button and then determining the object trajectory. Two methods can be used to determine the trajectory : mouse movement and loading the predefined trajectories. By mouse, the trajectory is simply given by moving the cursor while pressing the left button. The predefined trajectories are chosen through a menu.

Learning Termination

The Restart button is used to reinitiate the world without reinitiating the neural network weights. Basically the Restart button terminates the learning execution. If it is followed by clicking the Exit button, then the complete program would stop and exit. The Save Weight button can be used before exiting the program to save the resulted weight matrix and the tracking error history.

C.3 Future Developments

Some developments are still needed. A meaningful improvement could be made in the way of defining the simulation set-up parameters. The definition of the networks' size, learning parameter, and noise strategy could be improved by using a dialog window. In this way, the change of the definition in the middle of the execution is made possible. Another significant improvement is in dealing with network topology. At the moment, there is no interactive facility to define and choose the states from the world to be introduced as the neural network input. A graphical editor could be a solution for defining the neural network input scheme. This graphical editor can enhance the flexibility of the simulator in the term of defining the overall neural network architecture.

Appendix D

Robot Specifications

This appendix presents the kinematical specifications of the robot manipulators used either in the simulation or in the real implementation. These specifications are made according to the Denavit-Hartenberg convention.

D.1 American Robot AR 6500

The configuration of the AR 6500 manipulator is depicted as follows :

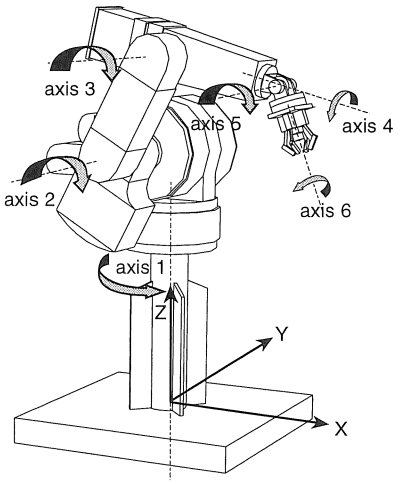


Figure D.1: Axes of the American Robot AR 6500 and its possible joint motions. Arrows indicate the motion positive directions.

Its kinematical specification is described as follows :

Link	a_{i-1} (mm)	α_{i-1} (deg)	d_i (mm)	θ_i (deg)
1	0	180	-1180	θ_1
2	0	90	0	$\theta_2 - 90$
3	440	0	0	$\theta_3 + 90$
4	0	-90	-440	θ_4
5	0	90	0	θ_5
6	0	-90	0	θ_6
camera	0	180	220	0

Table D.1: Denavit-Hartenberg parameters of the American Robot AR 6500.

To perform the specified tracking task, as a function of the velocities of the controlled joints 1 and 2, the velocity of each joint can be expressed as follows :

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \frac{d(\pi-(\int \dot{\theta}_2 dt + cos^{-1}((L_2 \cdot cos(\int \dot{\theta}_2 dt) + L_1 - L_6 - H)/L_3)))}{dt} \\ 0 \\ \frac{d(\pi-(\int \dot{\theta}_2 dt + \int \dot{\theta}_3 dt))}{dt} \\ -\dot{\theta}_1 \end{bmatrix} \tag{D.1}$$

where L_n is the length of link n, which from the Denavit-Hartenberg parameters specified above, can be rewritten as follows :

L_1	L_2	L_3	L_4	L_5	L_6 (+ camera)	(mm)
1180	440	440	0	0	220	

Table D.2: The lengths of the links of the American Robot AR 6500.

H is the camera height measured from the base frame of the robot (which is attached on the ground). This height is in fact the total of the height of the horizontal plane to the ground and the height of the camera to the horizontal plane.

D.2 Kuka 160 IR

The configuration of the Kuka 160 IR manipulator is depicted as follows :

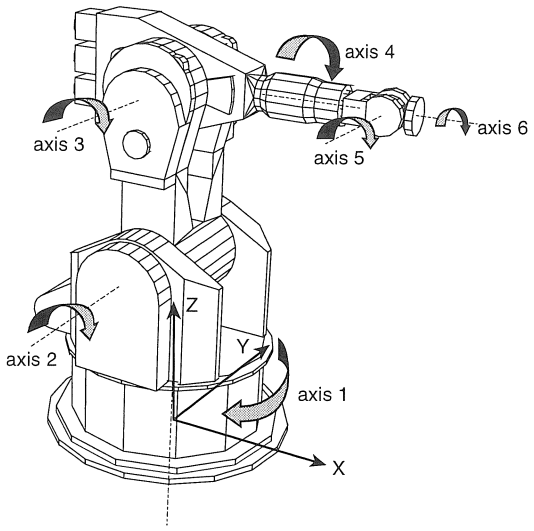


Figure D.2: Axes of the Kuka 160 IR robot and its possible joint motions. Arrows indicate the motion positive directions.

Its kinematical specification is described as follows :

Link	a_{i-1} (mm)	α_{i-1} (deg)	d_i (mm)	θ_i (deg)
1	0	180	-900	θ_1
2	0	90	0	$\theta_2 - 90$
3	970	0	0	$\theta_3 + 90$
4	0	-90	-1080	θ_4
5	0	90	0	θ_5
6	0	-90	0	θ_6
end-effector	0	180	140	0

Table D.3: Denavit-Hartenberg parameters of the Kuka 160 IR robot.

D.3 Kuka 361 IR

The configuration of the Kuka 361 IR manipulator is depicted as follows :

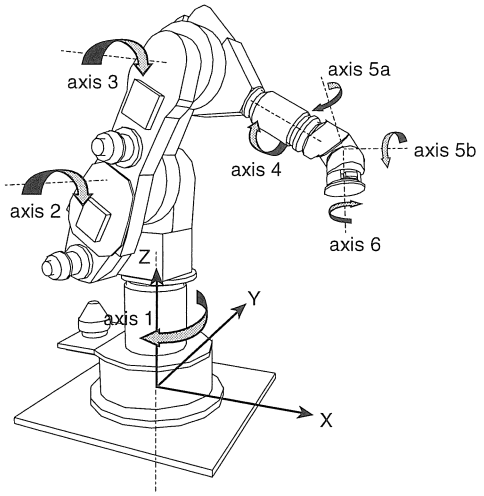


Figure D.3: Axes of the Kuka 361 IR robot and its possible joint motions. Arrows indicate the motion positive directions.

Its kinematical specification is described as follows :

Link	a_{i-1} (mm)	α_{i-1} (deg)	d_i (mm)	θ_i (deg)
1	0	180	-1050	θ_1
2	0	90	0	$\theta_2 - 90$
3	480	0	0	$\theta_3 + 90$
4	0	-90	-645	$\theta_4 + 90$
5a	0	-30	0	θ_5
5b	0	-120	0	θ_5
6	0	150	0	θ_6
end-effector	0	180	120	90

Table D.4: Denavit-Hartenberg parameters of the Kuka 361 IR robot.

Appendix E

Real Time Visual Object Localisation Technique

This appendix treats the technique applied in the learning visual tracking control implementation described in chapter 5. This technique is used to obtain the real-time visual information of object location in the camera field. The implementation of this technique is programmed by using C-language.

E.1 Introduction

The information about the position of the tracked object in the image frame captured by the vision system is obtained by implementing a 2-dimensional object localisation technique in real time. Two transputers are used to encourage the real-time performance. One transputer handles the image frame grabbing task while the other one handles the computation of the object localisation. A CCD camera with a resolution of 768×576 pixels is used.

E.2 Object Localisation Algorithm

The object position coordinate is determined by calculating the object's centre of gravity. In this implementation, the calculation of the centre of gravity is reduced to a 2-dimensional analysis which uses binary image as input. The binary image is obtained by applying initial image thresholding on the grey-scale image grabbed by the vision system. A pixel which exceeds the threshold is detected as an active pixel. If 2-dimensional analysis is applied, then the feature which is significant for calculating the centre of gravity is the area of

the object. The object area is defined as the number of pixels comprising the object multiplied by the area of a single pixel (frequently assumed to be a single unit). Practically, the object area can be simply calculated as the number of active pixels which appear after thresholding. An appropriate threshold value has to be chosen, which depends on the lighting condition and the opening of the camera's diaphragm. A too low threshold value gives much noise in the image and a too high threshold value produces a distorted image.

The 2-dimensional analysis method used for calculating the centre of gravity is described by the following equations :

$$X_c = \frac{\Sigma x_n}{A} \tag{E.1}$$

$$Y_c = \frac{\Sigma y_n}{A} \tag{E.2}$$

where X_c, Y_c = the location of the centre of gravity,
x_n = column coordinate of the n^{th} active pixel,
y_n = row coordinate of the n^{th} active pixel,
A = object area.

Figure E.1 shows the result of the applied object localisation algorithm.

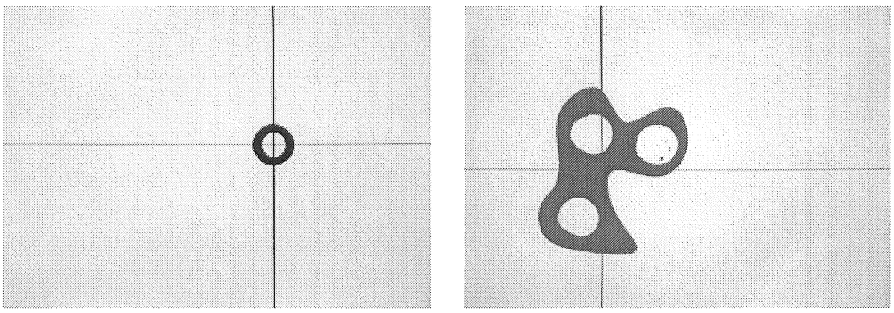


Figure E.1: The cross hair cursors indicate the computed centre of gravity locations of two example object shapes shown by their negative images.

E.3 Real-Time Implementation Strategy

Dealing with images implies dealing with a huge number of numerical values. A strategy is demanded to allow the available hardware coping with such

task. Two ways are followed to achieve the real time performance. Firstly, sharing and distributing the computation task to more than one transputer provides a chance to shorten the needed computing time. Secondly, reducing the size of the image obviously demands less computing time. A smaller size of image is achieved by transforming the grabbed image to a lower resolution image by software.

E.3.1 Task sharing between transputers

Both the frame grabbing process and the calculation of the centre of gravity takes some computation time, which may cause a certain delay. The idea applied to reduce the needed computing time is by distributing the tasks to two different transputers. This strategy would avoid one task to be executed after the other, serially on one processor.

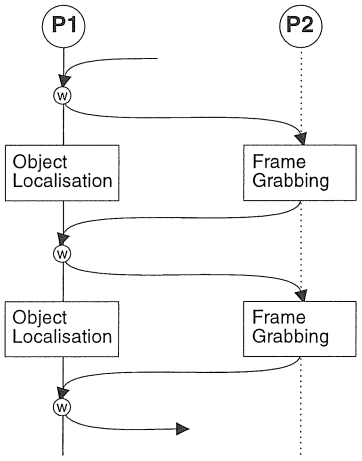


Figure E.2: The scheme of the computational task sharing between two transputers.

In this implementation, the frame grabbing task stays to be executed on the transputer 2 which is associated with the vision system, and the centre of gravity calculation is assigned to the transputer 1 (see figure 5.22).

This strategy allows both tasks to be executed parallelly. Figure E.2 shows that parallelism. P1 and P2 represent two processes running on transputers 1 and 2. Once a new image grabbed by transputer 2, it would be sent to transputer 1. Transputer 1 would be carrying out the centre of gravity calculation while transputer 2 grabbing the next image.

E.3.2 Image size reduction

Still, image size significantly influences the execution speed achieved by above parallelism. The centre of gravity calculation time certainly depends on the number of image pixels which have to be taken to account. Another aspect, the data transmission speed from transputer 2 to transputer 1, has also to be considered.

It is obvious that reducing the image size would give a better effect to above aspects. In this implementation, several sizes of images are tried out. The size reduction technique is applied by taking one pixel value to represent its neighbourhood. The representative pixel is taken linearly. It means that for an image size of $1/4$, the significant pixel value is taken every 4 steps of the reading iteration. In this way, the other three pixels are neglected.

The size reduction proces is carried out on the transputer 2. In this case, transputer 2 would send only the size-reduced image to transputer 1. It implies that a shorter transmission time is needed.

E.3.3 Real time performance

An experiment has been done to evaluate the localisation accuracy and the related computing time of several different sizes of input image. The image size of 1 (full image), $1/2$, $1/4$, $1/8$, $1/16$ and $1/32$ are tried out. A flat washer with a diameter of 12 mm is used as the localised object. The object is put at a distance of 50 cm from the camera. The experiment uses a camera lens with 6 mm focus length.

Image size	σ (in pixel unit)		Localisation rate (Hz)
	x position	y position	
full	0.2604	0.1396	0.3792
$1/2$	0.3513	0.2220	0.7299
$1/4$	0.3715	0.1777	1.3004
$1/8$	0.4986	0.2124	2.2351
$1/16$	0.5291	0.2198	3.6127
$1/32$	0.6927	0.2124	5.0302

Table E.1: Object localisation accuracy and its computation rate, performed by using different sizes of input images.

For the chosen experimental condition, the system can not localise the object anymore if the image size is lower than $1/32$. Table 1 shows the experimental result. The accuracy is measured by calculating the standard deviation of the estimated object position taken from 20 samples. A more evident comparison is presented by figure E.3.

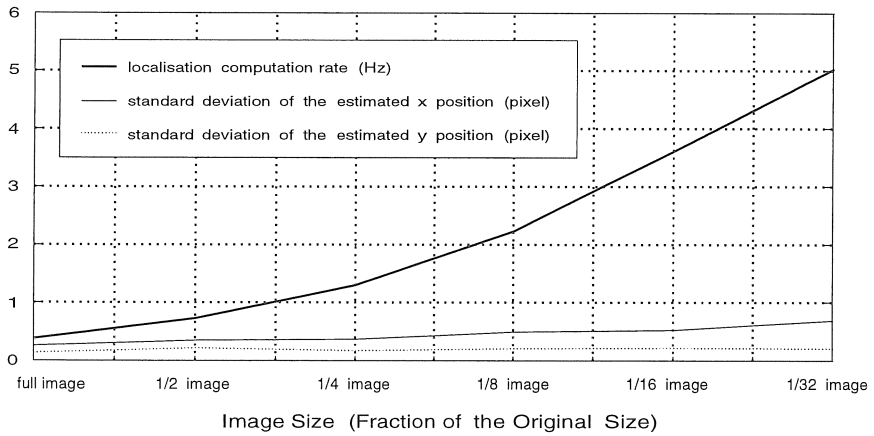


Figure E.3: The rate of the object localisation computation as a function of image size.

By reducing the image size, the number of frames that can be grabbed per second reasonably increases. It is seen that the standard deviation of the estimated y position is not affected by the reducing image size, while the estimated x position is. This condition occurs since the image size reduction yields a sparse representation of the object image in X-axis direction. It does not affect the object representation in Y-axis direction.

Bibliography

[Agre and Chapman, 1990] Agre, P. and Chapman, D. (1990). What are plans for ? In Maes, P., editor, *Designing Autonomous Agents*, pages 17–34. MIT Press, Cambridge, Massachusetts.

[Aleksander and Morton, 1993] Aleksander, I. and Morton, H. (1993). *Neurons and Symbols : The Stuff that Mind is Made of*. Chapman and Hall, London.

[Anderson, 1987] Anderson, C. W. (1987). Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, Irvine, California.

[Anderson, 1983] Anderson, J. A. (1983). Cognitive and psychological computation with neural models. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):799–815.

[Andrews, 1972] Andrews, H. C. (1972). *Introduction to Mathematical Techniques in Pattern Recognition*. John Wiley & Sons, New York, New York.

[Antsaklis, 1992] Antsaklis, P. J. (1992). Neural networks in control systems. *IEEE Control Systems Magazine*, April:8–10.

[Arbib, 1989] Arbib, M. A. (1989). Interacting subsystems for depth perception and detour behavior. In Arbib, M. A. and Amari, S. I., editors, *Dynamic Interactions in Neural Networks : Models and Data*. Springer Verlag, New York, New York.

[Arbib and House, 1987] Arbib, M. A. and House, D. H. (1987). Depth and detours : An essay on visually guided behavior. In Arbib, M. A. and Hanson, R. A., editors, *Vision, Brain and Cooperative Computation*. MIT Press/Bradford Books, Cambridge, Massachusetts.

[Arkin, 1995] Arkin, R. C. (1995). Intelligent robotic systems. *IEEE Expert Magazine*, April:6–8.

[Arthaya, 1995] Arthaya, B. M. (1995). *A Study on Task Specification and Control for Two Cooperating Robot Arms.* Ph.D. thesis, Faculty of Applied Sciences, K. U. Leuven, Leuven.

[Asada, 1990] Asada, H. (1990). Teaching and learning of compliance using neural nets : Representation and generalisation of nonlinear compliance. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1237–1244, Cincinnati, Ohio.

[Asada and Liu, 1991] Asada, H. and Liu, S. (1991). Transfer of human skills to neural net robot controllers. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 2442–2448, Sacramento, California.

[Asada and Yang, 1989] Asada, H. and Yang, B.-H. (1989). Skill acquisition from human experts through pattern processing of teaching data. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 1302–1307, Scottsdale, Arizona.

[Asada and Yang, 1992] Asada, H. and Yang, B.-H. (1992). Hybrid linguistic/numeric control of deburring robots based on human skills. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1467–1474, Nice.

[Asama, 1995] Asama, H. (1995). Cooperative action of robotic agents : Strategies and experiments using mobile robots. In *Tutorial Note on Task Oriented Multi Agent Robot Systems, The Fourth International Conference on Intelligent Autonomous Systems*, pages 9–16, Karlsruhe, Germany.

[Bajcsy, 1993] Bajcsy, R. (1993). Cooperative agents : Machines and humans. In *Proceedings of the International Conference on Advance Robotics*, pages 115–119, Tokyo.

[Barr and Kiernan, 1988] Barr, M. L. and Kiernan, J. A. (1988). *The Human Nervous System : An Anatomical Viewpoint.* J.B. Lippincott Company, Philadelphia, Pennsylvania.

[Barto and Anandan, 1985] Barto, A. G. and Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, man, and Cybernetics*, 15:360–375.

[Barto and Sutton, 1981] Barto, A. G. and Sutton, R. S. (1981). Landmark learning : An illustration of associative search. *Biological Cybernetics*, 42:1–8.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846.

[Barto et al., 1981] Barto, A. G., Sutton, R. S., and Brouwer, P. S. (1981). Associative search network : A reinforcement learning associative memory. *Biological Cybernetics*, 40:201–211.

[Barto et al., 1991] Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1991). Learning and sequential decision making. In Gabriel, M. and Moore, J. W., editors, *Learning and Computational Neuroscience*. MIT Press, Cambridge, Massachusetts.

[Beer, 1990] Beer, R. D. (1990). *Intelligence as Adaptive Behavior : An Experiment in Computational Neuroethology*. Academic Press, San Diego, California.

[Berns et al., 1991] Berns, K., Dillmann, R., and Hofstetter, R. (1991). An application of a backpropagation network for the control of a tracking behavior. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 2426–2431, Sacramento, California.

[Berns et al., 1992] Berns, K., Dillmann, R., and Zachmann, U. (1992). Reinforcement-learning for the control of an autonomous mobile robot. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1808–1815, Raleigh, New Carolina.

[Bloom and Lazerson, 1988] Bloom, F. E. and Lazerson, A. (1988). *Brain, Mind, and Behavior*. W.H. Freeman & Co., New York, New York, 2nd edition.

[Boden, 1987] Boden, M. A. (1987). *Artificial Intelligence and Natural Man*. The MIT Press, London, 2nd edition.

[Brooks, 1983] Brooks, R. A. (1983). Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(3):190–196.

[Brooks, 1986] Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23.

[Brooks, 1990] Brooks, R. A. (1990). Elephants don't play chess. In Maes, P., editor, *Designing Autonomous Agents*, pages 3–15. MIT Press, Cambridge, Massachusetts.

[Brooks, 1991a] Brooks, R. A. (1991a). Intelligence without reason. A.I. Memo No. 1293, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

[Brooks, 1991b] Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.

[Brooks, 1991c] Brooks, R. A. (1991c). New approaches to robotics. *Science*, 253:1227–1232.

[Brooks and Mataric, 1993] Brooks, R. A. and Mataric, M. J. (1993). Real robots, real learning problems. In Connel, J. H. and Mahadevan, S., editors, *Robot Learning*, pages 193–213. Kluwer Academic Publishers, Norwell, Massachusetts.

[Brooks and Stein, 1993] Brooks, R. A. and Stein, L. A. (1993). Building brains for bodies. A.I. Memo No. 1439, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

[Bruyninckx, 1995] Bruyninckx, H. (1995). *Kinematic Models for Robot Compliant Motion with Identification of Uncertainties*. Ph.D. thesis, Faculty of Applied Sciences, K. U. Leuven, Leuven.

[Camhi, 1980] Camhi, J. M. (1980). The escape system of the cockroach. *Scientific American*, 243(6):144–156.

[Carbonell et al., 1984] Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. (1984). An overview of machine learning. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning : An Artificial Intelligence Approach*, pages 3–23. Springer-Verlag, Heidelberg, Germany.

[Chandrasekaran et al., 1988] Chandrasekaran, B., Goel, A., and Allemang, D. (1988). Connectionism and information-processing abstractions : The message still counts more than the medium. *AAAI AI Magazine*, 9(4):25–34.

[Chen et al., 1991] Chen, S., Cowan, C. F. N., and Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309.

[Chen and Pao, 1989] Chen, V. C. and Pao, Y.-H. (1989). Learning control with neural networks. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 1448–1453, Scottsdale, Arizona.

[Connell, 1990] Connell, J. (1990). *Minimalist Mobile Robotics : A Colony-style Architecture for an Artificial Creature*. Academic Press, San Diego, California.

[Connell and Mahadevan, 1993] Connell, J. H. and Mahadevan, S. (1993). Introduction to robot learning. In Connel, J. H. and Mahadevan, S., editors, *Robot Learning*, pages 1–17. Kluwer Academic Publishers, Norwell, Massachusetts.

[Cooperstock and Milios, 1992] Cooperstock, J. R. and Milios, E. E. (1992). Adaptive neural networks for vision-guided position control of a robot arm. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 397–403, Glasgow.

[Davis, 1991] Davis, P. F. (1991). Orientation-independent recognition of chrysanthemum nodes by an artificial neural network. *Computers and Electronics in Agriculture*, 5:305–314.

[de Callataÿ, 1986] de Callataÿ, A. M. (1986). *Natural and Artificial Intelligence*. Elsevier Science Publishers, Amsterdam, The Netherlands.

[De Schutter, 1986] De Schutter, J. (1986). *Compliant Robot Motion : Task Formulation and Control*. Ph.D. thesis, Faculty of Applied Sciences, K. U. Leuven, Leuven.

[Deutsch and Deutcsh, 1993] Deutsch, S. and Deutcsh, A. (1993). *Understanding the Nervous System : An Engineering Perspective*. IEEE Press, New York, New York.

[Dillmann, 1994] Dillmann, R. (1994). Application of machine learning methodologies in robotics. In *Tutorial Note on Learning Technique in Robotics, Part 1, Technology Transfer Workshop on Industrial Vision and Autonomous Robots (IVAR'94)*, Leuven, Belgium.

[Dreyfus, 1979] Dreyfus, H. L. (1979). *What Computers Can't Do : The Limits of Artificial Intelligence*. Harper and Row, New York, New York, revised edition.

[Dubrawski and Crowley, 1994] Dubrawski, A. and Crowley, J. L. (1994). Self-supervised neural system for reactive navigation. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2076–2081, San Diego, California.

[Erdmann, 1986] Erdmann, M. (1986). Using backprojections for fine motion planning with uncertainty. *The International Journal of Robotics Research*, 5(1):19–45.

[Flynn, 1985] Flynn, A. M. (1985). Redundant sensors for mobile robot navigation. AI-TR-859, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

[Freeman and Skapura, 1991] Freeman, J. A. and Skapura, D. M. (1991). *Neural Networks : Algorithms, Applications, and Programming Techniques*. Addison-Wesley, Reading, Massachusetts.

[Fujii and Kurono, 1975] Fujii, S. and Kurono, S. (1975). Co-ordinated computer control of a pair of manipulators. In *Proceedings of the Fourth World Congress on the Theory of Machines and Mechanisms*, pages 411–417, Newcastle upon Tyne, United Kingdom.

[Fukuda and Shibata, 1990] Fukuda, T. and Shibata, T. (1990). Neural network applications for robotic motion control. In *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, pages SL31–SL36, Istambul.

[Fukuda and Shibata, 1992] Fukuda, T. and Shibata, T. (1992). Theory and applications of neural networks for industrial control systems. *IEEE Transactions on Industrial Electronics*, 39(6):472–489.

[Fukuda and Ueyama, 1994] Fukuda, T. and Ueyama, T. (1994). *Cellular Robotics and Micro Robotic Systems*. World Scientific, Singapore.

[Geschwind, 1986] Geschwind, N. (1986). Specializations of the human brain. In *Readings from Scientific American : Language, Writing, and the Computer*, pages 1–16. W. H. Freeman & Co., New York, New York.

[Giralt, 1984] Giralt, G. (1984). Mobile robots. *Robotics and Artificial Intelligence - NATO ASI Series*, F11:365–393.

[Gorman and Sejnowski, 1988] Gorman, R. M. and Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89.

[Grossberg, 1987] Grossberg, S. (1987). Competitive learning : from interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63.

[Grossberg and Kuperstein, 1989] Grossberg, S. and Kuperstein, M. (1989). *Neural Dynamics of Adaptive Sensory-Motor Control*. Pergamon Press, New York, New York.

[Gullapalli et al., 1992] Gullapalli, V., Grupen, R. A., and Barto, A. G. (1992). Learning reactive admittance control. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1475–1480, Nice.

[Gustavson, 1985] Gustavson, R. E. (1985). A theory for the three-dimensional mating of chamfered cylindrical parts. *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, 107(March):112–122.

[Handelman et al., 1989] Handelman, D. A., Lane, S. H., and Gelfand, J. J. (1989). Integrating neural networks and knowledge-based systems for robotic control. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 1454–1460, Scottsdale, Arizona.

[Harvey, 1994] Harvey, R. L. (1994). *Neural Network Principles*. Prentice Hall, Englewood Cliffs, New Jersey.

[Hashimoto et al., 1992a] Hashimoto, H., Kubota, T., Kudou, M., and Harashima, F. (1992a). Self-organizing visual servo system based on neural networks. *IEEE Control Systems Magazine*, April:31–36.

[Hashimoto et al., 1992b] Hashimoto, H., Kubota, T., Sato, M., and Harashima, F. (1992b). Visual control of robotic manipulator based on neural networks. *IEEE Transactions on Industrial Electronics*, 39(6):490–496.

[Hayati, 1986] Hayati, S. (1986). Hybrid position/force control of multi-arm cooperating robots. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 82–89, San Fransisco, California.

[Hecht-Nielsen, 1990] Hecht-Nielsen, R. (1990). *Neurocomputing*. Addison-Wesley, Reading, Massachusetts.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California.

[Hetherington and Parke, 1986] Hetherington, E. M. and Parke, R. D. (1986). *Child Psychology : A Contemporary Viewpoint*. McGraw-Hill, New York, New York, 3rd edition.

[Hildreth and Hollerbach, 1985] Hildreth, E. C. and Hollerbach, J. M. (1985). The computational approach to vision and motor control. A.I. Memo No. 846 – C.B.I.P. Memo No. 014, MIT Artificial Intelligence Laboratory and Center for Biological Information Processing, Cambridge, Massachusetts.

[Hoffman, 1986] Hoffman, J. E. (1986). The psychology of perception. In LeDoux, J. E. and Hirst, W., editors, *Mind and Brain : Dialogues in cognitive neuroscience*, pages 7–32. Cambridge University Press, New York, New York, 1st edition.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[Huang and Lippmann, 1987] Huang, W. Y. and Lippmann, R. P. (1987). Comparison between neural net and conventional classifiers. In *Proceedings of the IEEE First International Conference on Neural Networks*, pages 485–493, San Diego, California.

[Ichikawa and Sawa, 1992] Ichikawa, Y. and Sawa, T. (1992). Neural network application for direct feedback controllers. *IEEE Transactions on Neural Networks*, 3(2):224–231.

[Ishida, 1977] Ishida, T. (1977). Force control in coordination of two arms. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence - IJCAI'77*, pages 717–722, Cambridge, Massachusetts.

[Ito, 1984] Ito, M. (1984). *The Cerebellum and Neural Control*. Raven Press, New York, New York.

[Johnson-Laird, 1993] Johnson-Laird, P. (1993). *The Computer and the Mind : An Introduction to Cognitive Science*. Fontana Press, London.

[Kaelbling, 1993] Kaelbling, L. P. (1993). *Learning in Embedded Systems*. The MIT Press, Cambridge, Massachusetts.

[Kaelbling and Rosenschein, 1990] Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. In Maes, P., editor, *Designing Autonomous Agents*, pages 35–48. MIT Press, Cambridge, Massachusetts.

[Katupitiya, 1985] Katupitiya, J. (1985). *Vision Assisted Tracking and Time Optimal Acquisition of Moving Objects Using A Manipulator*. Ph.D. thesis, Faculty of Applied Sciences, K. U. Leuven, Leuven.

[Kawato et al., 1987] Kawato, M., Furukawa, K., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185.

[Kent, 1981] Kent, E. W. (1981). *The Brains of Men and Machines*. McGraw-Hill, New York, New York.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98.

[Khatib and Burdick, 1986] Khatib, O. and Burdick, J. (1986). Motion and force control of robot manipulators. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 1381–1386, San Francisco, California.

[Khotanzad and Lu, 1991] Khotanzad, A. and Lu, J. (1991). Shape and texture recognition by a neural network. In Sethi, I. K. and Jain, A. K., editors, *Artificial Neural Networks and Statistical Pattern Recognition*, pages 109–131. Elsevier Science Publishers, Amsterdam.

[Kimble, 1988] Kimble, D. P. (1988). *Biological Psychology*. Holt, Rinehart and Winston, New York, New York.

[Kohonen, 1987] Kohonen, T. (1987). *Self-Organizing and Associative Memory*. Springer-Verlag, Berlin, 2nd edition.

[Kohonen, 1988] Kohonen, T. (1988). An introduction to neural computing. *Neural Networks*, 1:3–16.

[Kröse et al., 1990] Kröse, B. J. A., van der Korst, M. J., and Groen, F. C. A. (1990). Learning strategies for a vision based neural controller for a robot arm. In *Proceedings of IEEE International Workshop on Intelligent Motor Control*, pages 199–203, Istambul.

[Kuncheva, 1988] Kuncheva, R. A. (1988). Pattern recognition method using two-dimensional hadamard transform. In *Proceedings of the International Conference on Robot Vision and Sensory Controls*, pages 231–236, Zurich, Switzerland.

[Kung and Hwang, 1989] Kung, S.-Y. and Hwang, J.-N. (1989). Neural network architectures for robotic applications. *IEEE Transactions on Robotics and Automation*, 5(5):641–657.

[Kuperstein, 1989] Kuperstein, M. (1989). Infant robot learns adaptive coordination. *VISION*, 6(3):8–9.

[Langton, 1989] Langton, C. G. (1989). *Artificial Life*. Addison-Wesley, Redwood City, California.

[Lee et al., 1992] Lee, S., Park, J., and Shimoji, S. (1992). Neurobotics : A new neural net approach to robot 3d perception and visuo-motor coordination. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 554–559, Raleigh, New Carolina.

[Lim and Chyung, 1987] Lim, J. and Chyung, D. H. (1987). A control scheme for two cooperating robot arms. *IEEE Control Systems Magazine*, February:65–68.

[Lippmann, 1987] Lippmann, R. P. (1987). An introduction to computing with neural nets. *Acoustics, Speech and Signal Proceesing Magazine*, 4(2):4–22.

[Lozano-Pérez, 1981] Lozano-Pérez, T. (1981). Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(10):681–698.

[Lozano-Pérez et al., 1984] Lozano-Pérez, T., Mason, M. T., and Taylor, R. H. (1984). Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24.

[Luh and Zheng, 1986] Luh, J. Y. S. and Zheng, Y. F. (1986). An interactively hierarchical control scheme for two coordinating industrial robots. In *Proceedings of the 25th Conference on Decision and Control*, pages 1265–1266, Athens, Greece.

[Maes and Brooks, 1990] Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence - AAAI'90*, pages 796–802, Boston, Massachusetts.

[McClelland and Elman, 1986] McClelland, J. L. and Elman, J. (1986). Interactive process in speech perception : The TRACE model. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing, Vol. 2, Chap. 15*. The MIT Press, Cambridge, Massachusetts.

[Mel, 1990] Mel, B. W. (1990). *Connectionist Robot Motion Planning : A Neurally-Inspired Approach to Visually-Guided Reaching*. Academic Press, San Diego, California.

[Miller, III et al., 1990] Miller, III, W. T., Glanz, F. H., and Kraft, III, L. G. (1990). CMAC : An associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567.

[Mitchell, 1990] Mitchell, T. M. (1990). Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence - AAAI'90*, pages 1051–1058, Boston, Massachusetts.

[Monostori and Barschdorff, 1992] Monostori, L. and Barschdorff, D. (1992). Artificial neural networks in intelligent manufacturing. *Robotics & Computer-Integrated Manufacturing*, 9(6):421–437.

[Moody and Darken, 1989] Moody, J. and Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294.

[Nakamura, 1991] Nakamura, Y. (1991). *Advanced Robotics : Redundancy and Optimization*. Addison-Wesley, Menlo Park, California.

[Narendra and Mukhopadhyay, 1992] Narendra, K. S. and Mukhopadhyay, S. (1992). Intelligent control using neural networks. *IEEE Control Systems Magazine*, April:11–18.

[Narendra and Parthasarathy, 1990] Narendra, K. S. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27.

[Nevatia, 1982] Nevatia, R. (1982). *Machine Perception*. Prentice Hall, Englewood Cliffs, New Jersey.

[Nuttin et al., 1995] Nuttin, M., Van Brussel, H., Peirs, J., Soembagijo, A. S., and Sonck, S. (1995). Learning the peg-into-hole assembly operation with a connectionist reinforcement technique. In *Preprints of the Second International Workshop on Learning in Intelligent Manufacturing Systems*, pages 335–357, Budapest, Hungary.

[Nye, 1992] Nye, A. (1992). *Xlib Programming Manual : for Version 11 of the X Window System.* O'Reilly and Associates, Inc., Sebastopol, California.

[Penrose, 1995] Penrose, R. (1995). Must mathematical physics be reductionist ? In Cornwell, J., editor, *Nature's Imagination : The Frontiers of Scientific Vision*, pages 12–26. Oxford University Press, Oxford, United Kingdom.

[Pomerleau, 1989] Pomerleau, D. A. (1989). ALVIN : An autonomous land vehicle in a neural network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan Kaufmann, San Mateo, California.

[Poo et al., 1992] Poo, A. N., Jr., M. H. A., Teo, C. L., and Li, Q. (1992). Performance of a neuro-model-based robot controller : Adaptability and noise rejection. *Intelligent Systems Engineering*, Autumn:50–62.

[Psaltis et al., 1987] Psaltis, D., Sideris, A., and Yamamura, A. (1987). Neural controllers. In *Proceedings of the IEEE First International Conference on Neural Networks*, pages 551–558, San Diego.

[Rabello and Avula, 1992] Rabello, L. C. and Avula, X. J. R. (1992). Hierarchical neurociontroller architecture for robotic manipulation. *IEEE Control Systems Magazine*, April:37–41.

[Reynaerts, 1993] Reynaerts, D. (1993). Tactile sensing data interpretation for object manipulation. *Sensors and Actuators A : Physical*, A37–A38:243–246.

[Reynaerts, 1995] Reynaerts, D. (1995). *Control Methods and Actuation Technology for Whole-Hand Dexterous Manipulation.* Ph.D. thesis, Faculty of Applied Sciences, K. U. Leuven, Leuven.

[Rich and Knight, 1991] Rich, E. and Knight, K. (1991). *Artificial Intelligence.* McGraw-Hill, Singapore, 2nd edition.

[Ritter et al., 1992] Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps : An Introduction.* Addison-Wesley, Reading, Massachusetts.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing, Vol. 1, Chap. 8.* The MIT Press, Cambridge, Massachusetts.

[Russell, 1990] Russell, R. A. (1990). *Robot Tactile Sensing*. Prentice Hall, Sydney.

[Saerens and Soquet, 1991] Saerens, M. and Soquet, A. (1991). Neural controller based on back-propagation algorithm. *IEE Proceedings-F*, 138(1):55–62.

[Schmidt, 1985a] Schmidt, R. F. (1985a). Motor systems. In Schmidt, R. F., editor, *Fundamentals of Neurophysiology*, pages 155–200. Springer-Verlag, New York, New York, 3rd edition.

[Schmidt, 1985b] Schmidt, R. F. (1985b). The structure of the nervous system. In Schmidt, R. F., editor, *Fundamentals of Neurophysiology*, pages 1–18. Springer-Verlag, New York, New York, 3rd edition.

[Sethi, 1991] Sethi, I. K. (1991). Decision tree performance enhancement using an artificial neural network implementation. In Sethi, I. K. and Jain, A. K., editors, *Artificial Neural Networks and Statistical Pattern Recognition*, pages 71–88. Elsevier Science Publishers, Amsterdam.

[Shibata and Fukuda, 1993] Shibata, T. and Fukuda, T. (1993). Coordinative behavior in evolutionary multi-agent-robot system. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 448–453, Yokohama, Japan.

[Shoham et al., 1992] Shoham, M., Li, C. J., Hacham, Y., and Kreindler, E. (1992). Neural network control of robot arms. *Annals of the CIRP*, 41(1):407–410.

[Simon, 1981] Simon, H. A. (1981). *The Sciences of the Artificial*. The MIT Press, Cambridge, Massachusetts, 2nd edition.

[Simons et al., 1982] Simons, J., Van Brussel, H., Schutter, J. D., and Verhaert, J. (1982). A self-learning automaton with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control*, AC-27(5):1109–1113.

[Simpson, 1990] Simpson, P. K. (1990). *Artificial Neural Systems : Foundations, Paradigms, Applications, and Implementations*. Pergamon Press, Elmsford, New York.

[Smolensky, 1988] Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioural and Brain Sciences*, 11(1):1–23.

[Soembagijo et al., 1995] Soembagijo, A. S., Ruiz, A. R. J., Reynaerts, D., Van Brussel, H., Ruiz, R. C., and Pons, J. L. (1995). Non-feature based classification of tactile contacts in a gripper using neural networks. *Submitted to*

the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, Minnesota.

[Soembagijo and Van Brussel, 1992] Soembagijo, A. S. and Van Brussel, H. (1992). Deriving robotic manipulation force vectors from tactile sensing images by using a feedforward neural network with error back-propagation learning. In *Proceedings of the Fourth Conference of Indonesian Aerospace Students in Europe*, Imperial College, London.

[Soembagijo and Van Brussel, 1994] Soembagijo, A. S. and Van Brussel, H. (1994). Learning of robot tracking behaviour by neural networks. In *Proceedings of the 1994 Pacific Conference on Manufacturing*, Jakarta, Indonesia.

[Soembagijo and Van Brussel, 1995a] Soembagijo, A. S. and Van Brussel, H. (1995a). Kinematic mappings of two cooperating arms using neural networks. In *Preprints of the Second International Workshop on Learning in Intelligent Manufacturing Systems*, pages 373–388, Budapest, Hungary.

[Soembagijo and Van Brussel, 1995b] Soembagijo, A. S. and Van Brussel, H. (1995b). Robot visual tracking control using neural networks. In *Proceedings of the Fourth International Conference on Intelligent Autonomous Systems*, pages 562–568, Karlsruhe, Germany.

[Soembagijo and Van Brussel, 1995c] Soembagijo, A. S. and Van Brussel, H. (1995c). A simulation environment for adaptive learning control of a robot tracking behaviour. *Submitted to be published in the International Journal of Artificial Intelligence in Engineering.*

[Sontag, 1992] Sontag, E. D. (1992). Feedback stabilization using two-hidden-layer nets. *IEEE Transactions on Neural Networks*, 3(6).

[Steels, 1994] Steels, L. (1994). Building agents with autonomous behavior systems. In Steels, L. and Brooks, R., editors, *The 'Artificial Life' route to 'Artificial Intelligence'. Building Situated Embodied Agents*. Lawrence Erlbaum Associates, New Haven. Also available at http://arti.vub.ac.be/www/memos/publications.html.

[Steels, 1995] Steels, L. (1995). When are robots intelligent autonomous agents ? *Journal of Robotics and Autonomous Systems*. In Press. Also available at http://arti.vub.ac.be/www/steels/autsys.ps.

[Stein, 1986] Stein, J. F. (1986). Role of the cerebellum in the visual guidance of movement. *Nature*, 323:217–221.

[Sutton, 1988] Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44.

[Sutton, 1991] Sutton, R. S. (1991). Reinforcement learning architectures for animats. In *From Animals to Animats : Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, pages 288–296, Cambridge, Massachusetts.

[Sutton et al., 1992] Sutton, R. S., Barto, A. G., and Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, April:19–22.

[Taipale, 1993] Taipale, T. (1993). A behavior-based control system applied over multi-robot system. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1941–1943, Yokohama, Japan.

[Tank and Hopfield, 1987] Tank, D. W. and Hopfield, J. J. (1987). Concentrating information in time : Analog neural networks with applications to speech recognition problems. In *Proceedings of the IEEE First International Conference on Neural Networks*, pages 455–468, San Diego, California.

[Thornton, 1992] Thornton, C. J. (1992). *Techniques in Computational Learning : An Introduction*. Chapman and Hall, London.

[Torras et al., 1994] Torras, C., Venaille, C., and Wells, G. (1994). Application of neural networks to image-based control of robot arms. In *Proceedings of the Second IFAC Symposium on Intelligent Components and Instruments for Control Applications (SICICA '94*, pages 281–286.

[Torras, 1992] Torras, C. (1992). Symbolic planning versus neural control in robots. In Rudomin, P., Arbib, M., and Cerrantes, F., editors, *Natural and Artificial Intelligence : A Meeting Between Neuroscience and Artificial Intelligence*. Research Notes in Neural Computing, Vol. 4.

[Van Brussel, 1994] Van Brussel, H. (1994). Holonic manufacturing systems : The vision matching the problem. In *Proceedings of the First European Conference on Holonic Manufacturing Systems*, Hannover, Germany.

[Van Brussel, 1995] Van Brussel, H. (1995). "Navigation" issues in intelligent autonomous systems *(invited paper)* . In *Proceedings of the International Conference on Intelligent Autonomous Systems (IAS-4)*, pages 42–52, Karlsruhe, Germany.

[Van Brussel and Belien, 1986] Van Brussel, H. and Belien, H. (1986). A high resolution tactile sensor for part recognition. In *Proceedings of the Sixth International Conference on Robot Vision and Sensory Controls*, pages 49–59, Paris, France.

[Van Brussel et al., 1994] Van Brussel, H., Nuttin, M., Pan, M., Soembagijo, A. S., Vanherck, P., and Verbeure, B. (1994). Applications of neural networks in mechatronics. In *Tutorial Note of the Workshop on Neural Networks Applications in Finance and Industry, K. U. Leuven*, Leuven, Belgium.

[Van Brussel and Simons, 1979] Van Brussel, H. and Simons, J. (1979). Robot assembly by active force feedback accommodation. *Annals of the CIRP*, 28(1):397–401.

[Van de Poel et al., 1993] Van de Poel, P., Witvrouw, W., Bruyninckx, H., and De Schutter, J. (1993). An environment for developing and optimising compliant robot motion tasks. In *Proceedings of the Sixth International Conference on Advanced Robotics ('93 ICAR)*, pages 713–718, Tokyo, Japan.

[Vandewalle, 1993a] Vandewalle, J. (1993a). Artificial neural networks : Feedforward and recurrent neural networks. *Lecture note 2, Graduate School in Systems and Control on Neural Networks,* K. U. Leuven.

[Vandewalle, 1993b] Vandewalle, J. (1993b). Artificial neural networks : Principles, methods, applications, status of the technology, prospects. *Lecture note 1, Graduate School in Systems and Control on Neural Networks,* K. U. Leuven.

[Vandewalle, 1993c] Vandewalle, J. (1993c). Artificial neural networks : Self-organizing networks and cellular neural networks. *Lecture note 3, Graduate School in Systems and Control on Neural Networks,* K. U. Leuven.

[Vandorpe and Van Brussel, 1994] Vandorpe, J. and Van Brussel, H. (1994). A reflexive navigation algorithm for an autonomous mobile robot. In *Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 251–258, Las Vegas, Nevada.

[Waibel, 1989] Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46.

[Wallace et al., 1985] Wallace, R., Stentz, A., Thorpe, C., Moravec, H., Whittaker, W., and Kanade, T. (1985). First results in robot road-following. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence - IJCAI'85*, pages 1089–1095, Los Angeles, California.

[Watkins and Dayan, 1992] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

[Weiss, 1984] Weiss, L. E. (1984). *Dynamic Visual Servo Control of Robots : An Adaptive, Image-Based Approach*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[Werbos, 1990] Werbos, P. J. (1990). A menu of designs for reinforcement learning over time. In Miller, III, W. T., Sutton, R. S., and Werbos, P. J., editors, *Neural Networks for Control*, pages 67–95. The MIT Press, Cambridge, Massachusetts.

[Whitney, 1982] Whitney, D. E. (1982). Quasi-static assembly of compliantly supported rigid parts. *ASME Journal of Dynamic Systems, Measurement, and Control*, 104(March):65–77.

[Wijesoma, 1993] Wijesoma, S. W. (1993). Eye-to-hand coordination for vision-guided robot control applications. *The International Journal of Robotics Research*, 12(1):65–78.

[Williams and Peng, 1989] Williams, R. J. and Peng, J. (1989). Reinforcement learning algorithms as function optimizers. In *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C.

[Xu and Van Brussel, 1995] Xu, H. and Van Brussel, H. (1995). Neural avoidance behaviour of the mobile robot lias. In *Proceedings of the International Conference on Neural Networks and Their Applications*, Marseilles, France.

[Yabuta and Yamada, 1990] Yabuta, T. and Yamada, T. (1990). Possibility of neural networks controller for robot manipulators. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1686–1691, Cincinnati, Ohio.

[Yabuta and Yamada, 1991] Yabuta, T. and Yamada, T. (1991). Learning ability of neural network control. NTT Telecommunication Field Systems R&D Center, Tokai, Ibaraki.

[Zhang and Huang, 1995] Zhang, H. C. and Huang, S. H. (1995). Applications of neural networks in manufacturing : A state-of-the-art survey. *International Journal of Production Research*, 33(3):705–728.

[Zheng and Luh, 1985] Zheng, Y. F. and Luh, J. Y. S. (1985). Control of two coordinated robots in motion. In *Proceedings of the* 24^{th} *IEEE Conference on Decision and Control*, pages 1761–1765, Fort Lauderdale, Florida.

[Zurada, 1992] Zurada, J. M. (1992). *Introduction to Artificial Neural Systems*. West Publishing Company, St. Paul, Minnesota.